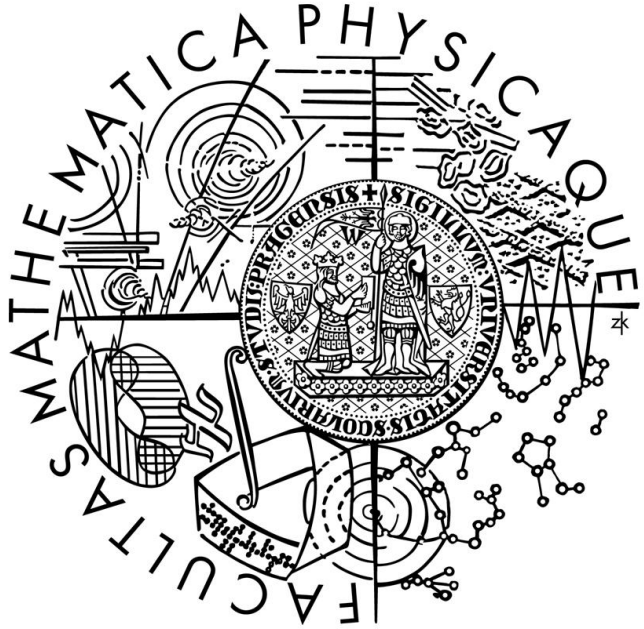
**Univerzita Karlova v Praze**
**Matematicko-fyzikální fakulta**

# DIPLOMOVÁ PRÁCE




Pavol Kováč

**Evaluation of methods for conceptual design and implementation of service-oriented applications based on enterprise integration systems**

Katedra softwarového inženýrství
Vedoucí diplomové práce: Ing.Lucia Kapová
Studijní program: Informatika

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne ..... .....

Pavol Kováč

# Table of Content

**Abstrakt**

Název práce: *Vyhodnocení metod konceptuálního designu a implementace servisně orientovaných aplikací založených na systémové integraci*.

Autor: *Pavol Kováč*
Katedra (ústav): *Katedra softwarového inženýrství*
Vedoucí diplomové práce: *Ing. Lucia Kapová*
E-mail vedoucího: *lucia.kapova@mff.cuni.cz*

Abstrakt: *Servisně orientované systémy (SOA) se stávají běžně používanými při budování rozlehlých aplikací. Cílem této diplomové práce je vyhodnocení metody konceptuálního návrhu a implementace servisně orientovaných systémů založených na podnikových integračních systémech. Detailně se zaměří na analýzu tří frameworků pro vývoj SOA aplikací a implementování případové studie v těchto frameworcích, následně pak budou vyhodnoceny možnosti koordinace služeb. Součástí práce by měla být i implementace případové studie a nastínění procesu vývoje SOA aplikací.*

Klíčová slova: *SOA, ESB, BPM, integrace*

**Abstract**

Title: *Evaluation of methods for conceptual design and implementation of service-oriented applications based on enterprise integration systems.*

Author: *Pavol Kovac*
Department: *Department of Software Engineering*
Supervisor: *Ing. Lucia Kapova*
Supervisor's e-mail address: *lucia.kapova@mff.cuni.cz*

Abstract: *Service-oriented development (SOA) has become commonly used approaches for building large systems. The goal of the thesis is to evaluate of methods for conceptual design and implementation of service-oriented applications based on enterprise integration systems. In more detail, the work should analyze three frameworks for SOA development, implement a case study in these systems and based on the analysis evaluate possibilities of service coordination. The implementation of case study and introduction of steps of SOA development process is expected.*

Keywords: *SOA, ESB, BPM, integration*

# 1. Introduction

In the life of every organization, which has at least more than one department, certain events may lead to the breaking point, in which the organization has to make a decision. Collaboration among departments is poor, in years, they have been implementing proprietary systems to serve their needs and now the things have become unbearable. The budget is tight and an urge to make cuts is emerging. Now it is the right time to enterprise integration.

The enterprise integration we are talking about might be perceived as an effort to achieve a certain level of cooperation of units (the departments in this case). What had been done formerly via internal mail service and printed forms, now it is possible to do electronically. Different internal systems virtually possessing the same functionality may be merged in a manner of having one façade to access all. All thanks to the enterprise integration. That is just the beginning of what the integration allows.
This organization having been depicted above needs not to be a company necessarily. A college itself may serve as an example as well. Consisting of departments, it is similar to a company in many ways. That means it can have the same integration need that company has.

This market need has been noticed by information companies, which identified the enterprise integration highly profitable. In spite of a lack of experience, the IT companies have made a number of integrations. On the other hand, these integrations were made in a proprietary way due to the lack of suitable standards for transportation, xml processing, format transportation and enterprise integration itself as well. Later after filling this gap, a few approaches started to dominate the field of enterprise integration. Service-oriented architecture (SOA), Enterprise integration bus (ESB) and business process management (BPM) are certainly worth mentioning.
Due to the long lasting gap in standard for enterprise integration, frameworks, which introduced their own approaches of enterprise integration, have become very popular and widely used for their capabilities. Their influence in this area cannot be simply ignored.

The topic of enterprise integration is vast. Many feasible approaches with large differences in vital characteristics cause headaches to decision makers. As clients looking for an integration solution are mostly corporations and large enterprises, every bad decision could cost a fortune. In worse case, it may be fatal. A simple overlooking of a lack of support in the matter of data transformation, for instance, may lead to an increase of expenses due to an unexpected need to create a custom remedy if possible.

Since the approaches for enterprise integration are very different in their nature and bring different views on enterprise integration, I decided to assess them by evaluating the most representative adoptions of theirs by using a real-life case study.

I decided to focus on ESB and BPM. ESB has a number of adoptions embodied in frameworks. BPM with its implementation jBPM by jBoss introduced a different view on enterprise integration. Large players for example, banks and insurance companies have successfully implemented this product with promising results. As a third approach, I decided to take the most popular framework and its proprietary concept.

The case study will be implemented once per each framework. A purpose of the case study is to provide basis upon which the framework can be evaluated systematically as per certain predetermined criteria. These criteria shall cover various aspects of a development, for example, code generation, tool support, deployment options, etc. The criteria will be imposed allowing those it may concern (system architects, decision makers, etc.) to make their own comparison respecting their actual needs.
My intent is not to compare the frameworks one another. They are not equivalent, because they represent different approaches. No comparison made would be objective in this case. I am not going to make performance tests either, for the same reason. Besides, the frameworks require different runtime environment, for example, they run on different application server, what eventually could make a significant impact in results.

In conclusion, I am going to make an overall assessment of the concepts the selected framework represents, where I will focus on determining the strong points and weaknesses. A recommendation regarding a use of concepts in the real-life would be part of the assessment.

## *1.1. Goals*

The main goal of this work is to evaluate concepts of enterprise service bus (ESB), business process management (BPM) and a concept of a popular proprietary framework.

Another goal is to introduce and describe selected frameworks from technical point of view and to outline steps of development process. This work is not intended to make a direct comparison of the used frameworks.

# 2. Background

## *2.1. What is System integration?*

System integration, also known as Enterprise Application Integration, is linking of component subsystems into one system and ensuring that the subsystems function together as a system. System integration is the process of linking different computing systems and software applications together physically or functionally. The system integrator brings discrete systems together utilizing a variety of techniques such as computer networking, enterprise application integration, business process management or manual programming. (Source [19]).

From now on, by using term of integration means system integration, for short.
This definition raises several questions. Who needs integration? Why to integrate? How to integrate? The last question is the topic of this work.
Other two questions are explained in next section.

### 2.1.1. System Integration example

System integration is tightly connected to the world of business for a couple of reasons. Firstly, companies in general often merge themselves or one take over another. Then system integration is the right solution to put this jungle of information systems into harmony. Secondly, system integration is expensive matter that requires many resources. By choosing the most appropriate approach of system integration, costs may be minimized.
One of the possible scenarios of system integration is to merge two (or more) information systems providing identical (or similar) functionality into one system, under one user interface. Consider following example:

Introduction:
In the beginning, there are two companies A and B. Both of them have functional internal information system to manage their employees. These information systems are identical in terms of functionality, but they have different user interface and underlying technologies.

Problem:
Suddenly company A buys company B. Company A realizes that aforementioned information systems are different and a need to system integration emerges.

Task (business point of view):
Integrate information systems. That means, to create an integration solution that would provide the same functionality as current information systems under single user interface.
Minimize overall cost, time, and used resources. Evaluate and reduce impact of modification of information systems. In other words, minimize possible dependences between those information systems.

Assumptions (technical point of view):

1. legacy information systems stay untouched
2. legacy information systems are not aware of one another
3. no existence of single point of failure
4. no dependency on operation system, application platform, etc
5. high scalability
6. centralized management
7. certain level of security
8. high availability

This list of assumption is extendable with many more. This warm-up example is supposed to show what a typical call for integration looks like. Integration is general term, which involves several different approaches. The next chapter deliberates a few classifications of integration.

### 2.1.2. Integration types

The primary target of integration is either to link systems within organization (in this case we talk about internal integration – Enterprise Application Integration - EAI) or to secure communication among systems of partner organizations (in this case we talk about external integration, Business-to-Business Integration - B2B). Requirements for these integration types are obviously different. Whereas by EAI, systems within organization are trustworthy and you might consider communication lines secure and reliable, by B2B there is no such guarantee. In addition, exposed interfaces by B2B must be reduced in order to prevent revealing confidential data. When creating B2B integration, the strongest available cipher method must be used, typically certificates, to secure communication line. These lines often involve internet, which definitely cannot be considered trustworthy and secure medium.

There is more than one approach for integration of applications. Each approach addresses some of the integration criteria better than others. The various approaches can be summed up in four main integration styles [2]:

1. File Transfer - have each application produce files of shared data for others to consume, and consume files that other have produced
2. Shared Database - have the applications store the data they wish to share in a common database
3. Remove Procedure Invocation - have each application expose some of its procedures so that they can be invoked remotely, and have applications invoke those to run behaviour and exchange data
4. Messaging - have each application connect to a common messaging system, and exchange data and invoke behaviour using messages

Every style has advantages and disadvantages. As you can see, the order reflects and increasing order of sophistication. Each style is looking for a more sophisticated approach to address the shortcomings of its predecessors. It is out of range of this work to take a detail look at the aforementioned integration styles. For more information, see source [2].

The remainder of this work will expand on messaging style of integration. I focus on messaging in part because we believe it is often the best style for solving many integration opportunities. It is also the least well understood of the integration styles. Finally, messaging is the basis for many EAI products and therefore it is good to know how it works.

### 2.1.3. Integration requirements

Integration itself is a difficult process. The dependent systems could be completely different; they could be built on specific platform (Java, .NET, CORBA ...) communicating using various protocols (TCP/IP, FTP, JSM, etc.). The main reason for integration is money. Companies are not willing to invest significant amount of money to have their information systems rebuilt from scratch. Integration seems to be a more feasible solution.

According to [2] integration criteria are:

1. Application Coupling
   Even integrated applications should minimize their dependencies on each other so that each can evolve without causing problems for the others. Tightly coupled applications make numerous assumptions about how the other applications work; when the applications change and break those assumptions, the integration breaks. The interface for integrating applications should be specific enough to implement useful functionality, but general enough to allow that implementation to change as needed.

2. Integration Simplicity
   When integrating an application into an enterprise, developers should strive to minimize changing the application and minimize the amount of integration code needed. Yet changes and new code will usually be necessary to provide good integration functionality, and the approaches with the least impact on the application may not provide the best integration into the enterprise.

3. Integration technology
   Different integration techniques require varying amounts of specialized software and hardware. These special tools may be expensive; it can lead to vendor lock-in, and increase the burden on developers to understand how to use the tools to integrate applications.

4. Data format
   Integrated applications must agree on the format of the data they exchange, or must have an intermediate translator to unify applications that insist on different data formats. A related issue is data format evolution and extensibility—how the format can change over time and how that will affect the applications.

5. Data timeliness
   Integration should minimize the length of time between when one application decides to share some data and other applications have that data. Data should be exchanged frequently in small chunks, rather than waiting to exchange a large set of unrelated items. Applications should be informed as soon as shared data is ready for consumption. Latency in data sharing has to be factored into the integration design; the longer sharing can take, the more opportunity for shared data to become stale, and the more complex integration becomes.

6. Data or functionality
   Integrated applications may not want to simply share data. They may wish to share functionality such that each application can invoke the functionality in the others. Invoking functionality remotely can be difficult to achieve, and even though it may seem the same as invoking local functionality, it works quite differently, with significant consequences for how well the integration works.

7. Asynchronicity
   Computer processing is typically synchronous, such that a procedure waits while its subprocedure executes. It is a given that the subprocedure is available when the procedure wants to invoke it. However, a procedure may not want to wait for the subprocedure to execute; it may want to invoke the subprocedure asynchronously, starting the subprocedure but then letting it execute in the background. This is especially true of integrated applications, where the remote application may not be running or the network may be unavailable—the source application may wish to simply make shared data available or log a request for a subprocedure call, but then go on to other work confident that the remote application will act sometime later.

These criteria are only the beginning. There are more fine-grained criteria, for example, for batch transfers (appropriate timing and time planning, acceptable latency, free space management ...).

### 2.1.4. Integration topologies

Designing of the integration is long and crucially important process. An integration architect has to take into consideration many aspects besides requirements themselves. It is not wise to underestimate this phase, because impacts may come back to you when you would not expect. Very good source of information is Microsoft Knowledge Base (source [20]).

System integration comes with several integration patterns.

1. point-to-point
2. broker
3. bus

These patterns come from practise. They have their cons and pros. Let us examine them step-by-step.

## 1. Point-to-point integration pattern

Many integration projects start with the need to connect two systems, and the easiest way to do so is to use the Point-to-Point Connection pattern. A point-to-point connection ensures that only one receiver receives a particular message. For this to work, the sending system must know the location of the receiving node. The sending system often must translate the message into a format that the receiving system understands.

When you use point-to-point connections, each system determines the address of all the other nodes that it needs to communicate with. When target addresses or protocol details change, all the systems that communicate with the target server must be updated. As the size of your integration network grows and as the frequency of change increases, the operational cost associated with this approach becomes significant.

Most integration scenarios require you to transform the data between the source system and the target systems. Additionally, in many scenarios, developers want to take advantage of some conditional logic when they configure message routing. If you use point-to-point connections, this logic is often duplicated on each server that requires transformation and routing. Duplicate code is expensive to write, and it is difficult to maintain, to extend, to test, and to manage.

The strength of the Point-to-Point Connection pattern is how simple it is to implement. The weakness of the Point-to-Point Connection pattern is the duplication of transformation and routing code between systems, and the high configuration cost of endpoint address changes. To minimize these weaknesses, you can add another layer of indirection between endpoints that contains a broker.

Consider point-to-point topology pattern, where every system is connected to each other. Evidently, this could lead to series of problems like difficult and expensive maintenance, high expenses for infrastructure, because point-to-point topology implies that number of edges equals to $n*(n-1)/2$, where $n$ is number of vertices. In addition, when you try to add another system you will have to add links to $(n-1)$ current systems.

However, this point-to-point pattern might of suitable for solution, when budget for integration is very small, there is no time to implement more sophisticated solution and number of systems converges to number two. Another condition supposed to be that there are no other integration plans for the future like for example, to incorporate a new external service into current integrated solution.

## 2. Broker integration pattern

Most of problems of the previous topology are solved with broker topology. A broker pattern is widely used and has many modifications. A common broker topology consists of centralized broker and (called hub) what particular information systems (called spokes) are connected using adapters. Adapters transform messages between broker and spokes; and broker is responsible for message delivery, routing and validation. Broker must be able to handle endpoint registration.

Whereas point-to-point topology has faced maintenance problems, broker topology must face the fact that as long as only one broker is used, it becomes a single point of failure. It means that if broker goes down, whole system goes down. There is a remedy for this called federated hub-and-spoke topology. This topology replaces single broker with many. If one broker breaks down, another one will takes his place. In spite of broker topology with single broker, configurations must be duplicated on every broker.

The broker pattern is used in Common Object Request Broker Architecture (CORBA), Microsoft Distributed Common Object Model (DCOM) or Universal Description Discovery and Integration (UDDI).

## 3. Message Bus integration pattern

Message bus is integration pattern analogous to an ordinary bus. A bus example you can find in your computer. There is a central bus, what every peripheral device is attached to. All communication between any two devices is routed via bus. It is possible to remove a certain device (like USB flash disk) without affecting entire set.

This principal works the same way for message bus as well. Message bus provides the highest level of application decoupling. It allows adding or removing application without affecting the others. Every application is connected to bus through adapter that works as a bridge between bus and external system.

The message bus pattern is the building rock of ESB, one of the more concrete concepts of SOA. This chapter deliberates system integration in general. To sum up, it introduced typical integration requirements, several topologies and a couple of different approaches toward integration. Next chapter is dedicated to a more specific topic of system integration, the service-oriented architecture (SOA) and a concrete application of message bus pattern, ESB.

## 2.2. Service-oriented Architecture (SOA)

Service Oriented Architecture (SOA) is a software architecture where functionality is grouped around business processes and packaged as interoperable services. SOA also describes IT infrastructure, which allows different applications to exchange data with one another as they participate in business processes. The aim is a loose coupling of services with operating systems, programming languages and other technologies, which underlie applications. SOA separates functions into distinct units, or services, which are made accessible over a network in order that they can be combined and reused in the production of business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services. SOA concepts are often seen as built upon and evolving from older concepts of distributed computing and modular programming.

It exists many definitions of SOA, the most quoted one is that created by OASIS (the Organization for the Advancement of Structured Information Standards) defining SOA as the following:

"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

I agree with this definition up to a point. Please notice that this SOA definition does not mention any messaging. This fact is important later in definition of term "Enterprise Service Bus". The definition does not actually specify the term of service, service description, etc. I provide alternative definition involving definition of useful terms, which I use in the rest of this work.

Let me define particular terms to clarify SOA step-by-step.

a. Service
   is a function that is well-defined, self-contained, and does not depend on the context or state of other services [22]. There are more definitions of term service, but this one is sufficient. Another definition is: services are implementation-independent reusable business functions that are discovered as self-describing interfaces, and invoked using open standard protocols across networks [23]. They are often implemented as web services.

b. Service Description
   is an agreed schema for describing what the service is, how it should be invoked, and what data is required to invoke the service successfully [22]. An example of service description language is Web Service Definition Language (WSDL).

c. Service Registry
   is a repository of service and data, which may be used by service providers to publish their interfaces, and service consumers to discover or find available services. The service registry may provide other functions to services that require a centralized repository [22]. A UDDI protocol is often used to construct a service registry.

d. Quality of Services
   Services have certain attributes that determine quality of service. For example, security, reliability, transactions, etc. Web services implement these attributes using protocols from family WS-*.

e. Business Process
   is a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement. Note that a business process could be considered a service in its own right, which leads to the idea that business process may be composed of services of different granularities [22].

The term of business process is the most important one. Realizing its substance, we can put together a set of services, under some conditions, to model a business process. To achieve this, I have to define a language that will manage service coordination. Furthermore, we need to define runtime environment for this language. A real-life example related to Enterprise Service Bus (ESB), being explained later in its own chapter, could be service coordination language BPEL and JBI as a runtime environment.

Having those terms defined, we can finally move to SOA definition in detail.

SOA is a concept that allows reusing heterogeneous legacy systems (services) with well-defined interfaces (service description) in a loosely coupled fashion to create a business process. Services are not aware of one another, but they are discoverable through service repository. Loose coupling secures that if one service changes, it will have minimal impact on other services.

## *2.3. Enterprise Service Bus (ESB)*

SOA itself is a high-level concept, the way of building application from small cells called services, which are most likely independently developed by potentially separate people. This concept is generic.
One of the more concrete concepts emerging from SOA is Enterprise Service Bus. It expands on SOA ideas into more specific pieces. It concerns architectural issues (bus), communication issues (messaging), service coordination (orchestration) and many more. It complements SOA definition with many recommendations. The list of potential recommendations and features of ESB is long and there is exists no consensus regarding them.

There is some disagreement on whether an enterprise service bus is an architectural style, a software product, or a group of software products. While use of an ESB certainly implies adherence to a particular architecture, the term "enterprise service bus" is almost always used to denote the software infrastructure that enables such an architecture.

A fact, that ESB has no widely-accepted definition enables companies to put suffix ESB to a product name to increase sales. The market is then full of pseudo ESB products and every company claims that its product is the true implementation of ESB principals. In this work, by using ESB I mean architectural style, to avoid any misunderstandings.

One of the first definitions of ESB (source [1]) comes from analyst of company Gartner Inc. Roy Schulte. He wrote in report in 2002 (DF-18-7304) the following:

"A new form of enterprise service bus (ESB) infrastructure – combining message-oriented middleware, web services, transformation and routing intelligence – will be running in the majority of enterprises by 2005.
These high-function, low-cost ESBs are well suited to be the backbone for service-oriented architectures and the enterprise nervous system."

Those four pillars: MOM, web services, transformation and routing intelligence represent the foundation of any good ESB. Web services are well-known technology nowadays. This definition of ESB recommends implementing services as a web services. Let me introduce other three with respect to ESB principals.

### 2.3.1. Message-oriented middleware (MOM)

MOM is a key part of ESB architecture. The source [3] defines MOM as a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces. APIs that extend across diverse platforms and networks are typically provided by the MOM.

Messages are autonomous units of transaction. They are the means of data transport. A message in terms of ESB and MOM consists of three parts: header, properties and body. Header is used to identify and route the message. Properties support application-specific values passed with the message, for instance, correlationID, MessageID or ReplyTo. Body of the message is actual payload. It should be XML. XML allows easily transforming and routing messages through the bus. ESB does not require usage of XML as a message payload, but it is strongly recommended due to reasons mentioned above.

MOM is software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism.

MOM comprises a category of inter-application communication software that generally relies on asynchronous message-passing, as opposed to a request/response metaphor.

Most message-oriented middleware depend on a message queue system, but there are some implementations that rely on broadcast or multicast messaging systems.

Usually MOM is implemented using JMS[1] or equivalent asynchronous messaging system. However, it is not a necessity. For example, WS (Web Services) standards like WS-ReliableMessaging and WS-Reliability are also suitable as MOM for ESB along with other WS* family protocols.

### 2.3.2. Transformation

I start with a simple example to introduce problems, which transformation solves.
Customer (retailer) wants to integrate his two enterprise information systems. The output of both systems is a sorted list of offered goods. Every item on this list has certain properties: name, count and price. Sorting rules are configurable.

The task is to integrate aforementioned systems so that the output will be a sorted list of goods, which consists of union of the list produced by those two systems.

You probably see a couple of pitfalls right about now.

a. What is the currency of result list? Is there a need to convert currency by any way?
b. What about duplicated items? Should they be aggregated by count?
c. Does rounding work the same way for both systems? What is a number representation? How many fraction digits are allowed in numbers?

If any of the bullets above is true, then transformation takes its place. ESB recommends using XML as a message format for a reason. In compare with native binary format of data, XML is human-readable and it is simple transformable. On the other hand, its shortcoming is its size and speed. XML makes additional unwanted overhead, which must be definitely taken into consideration during designing of the integration solution. Otherwise, solution might face performance problems. XML gathers technologies to format, transform and querying under the one roof. Namely, it is XSL, XQUERY, XPATH, etc. Usage of these technologies along with XML as a message format is one of the ESB recommendation.

[1] JMS stands for Java Message Service by Sun Microsystems

In the example above, a need to transformation emerges. Currencies and rounding have to be reunited and duplications have to be eliminated. If XML format is used, then XSLT transformation will be suitable to solve given issues.

Message transformation in terms of ESB is often implemented as a special service (service engine, when complying JBI standard as well). Its configurations requires XSLT template. It transforms, extracts, enriches or aggregate incoming data according to inputs and XSLT template.

Next part deliberates routing intelligence that is the fundamental of service coordination.

### 2.3.3. Routing intelligence

With the ESB, applications, services and components do not interact directly. The ESB serves as an intermediary for communication and the routing of messages between components. To the contrary, client-server architecture does routing in a tightly-coupled fashion. The client is aware of the server and vice versa. They need to find one another and invoke it.
Since components are not aware of each other, they still need to collaborate. ESB separates service lookups from the components that require the services. Therefore, developers spend less time implementing this code for every component in the system. This feature of ESB is call message routing. It allows you to define abstract message flows also known as workflows that represent a business process. With workflows, components can plug into at runtime dynamically. No hard-coding is needed.

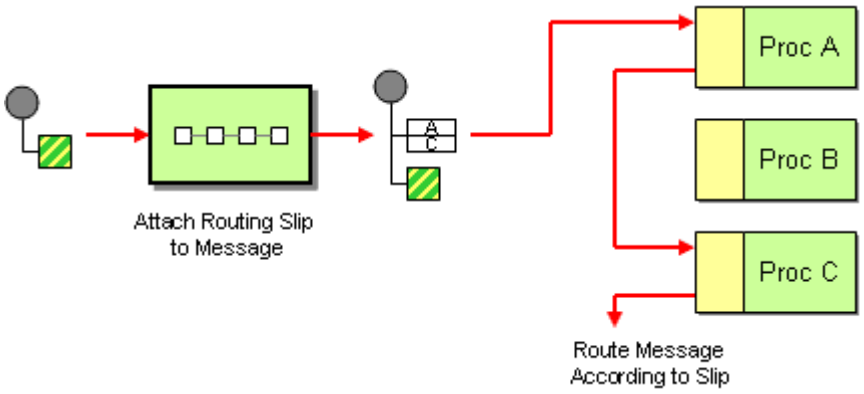
Prior to continue with routing I would like to define Abstract Service Endpoint.
A Service Endpoint is defined as a discrete operation devoid of its implementation [1]. Abstract service endpoint is an abstraction, which lies between the bus and a service. It comprises input endpoint, output endpoint, fault endpoint, endpoint for refused messages or monitoring endpoint.

Then addressing of services performs as follows. A service accepts a message from its input endpoint, then performs its logic and places one or more messages onto its output endpoint. For example, a routing service that routes basing on content of the messages (Content-based routing). After it receives message from its input endpoint, it reads the content and changes a destination (input endpoint of another service) in the header of the message. Afterwards it put the updated message into its output endpoint. The routing is then founded on the basis of a content of the message.

The routing intelligence can be done in either centralised or decentralized way. It differs in presence of a single routing service. Source [1] describes these two ways as a content-based routing and itinerary-based routing.

1. Itinerary-based routing

Itinerary-based routing comes out of an enterprise integration pattern (source [2]) called Routing Slip (Picture 2-1). With this approach, a workflow is stored in a message metadata usually in XML format. The message carries whole itinerary containing all steps of a certain use case. Itinerary-based routing is also known as choreography.



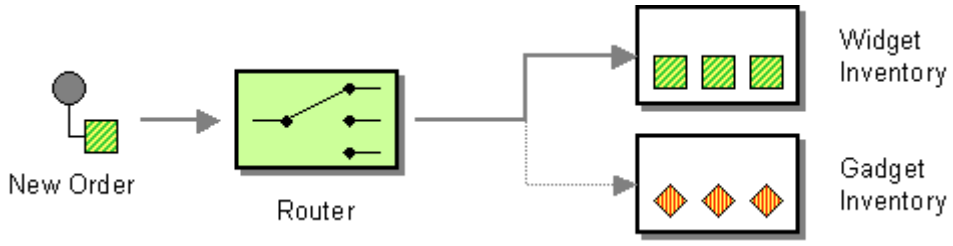

**Picture 2-1: Routing Slip pattern**

For example, before submitting an order, it is necessary to check the credit and store availability and then print a invoice. These steps are part of itinerary being stored in message metadata. An advantage is that a need for centralised management element is no longer relevant.

Itinerary-based routing is hard to implement in compare with content-based routing. It has not been widely supported yet by ESB framework producers. In fact, only Sonic has implemented itinerary-based routing by its nature. Sonic ESB likes to push itinerary-based routing as a core feature of its ESB (Source [24]). Microsoft also pushed this approach of itinerary-based routing with their WS-Routing specification. However, the standards process has jettisoned WS-Routing in favour of WS-Addressing. Itineraries are inherently insecure, because every ESB intermediary must modify the message header as it is going through its itinerary. Another withdraw of itineraries is that they are 100% proprietary. There is no standard on this field and probably it will never be.
A new standard emerges bringing a light into itinerary-based routing. Its name is Web Services Choreography Description Language (WS-CDL). It is has not been standardized by W3C yet. Now it is a W3C Candidate Recommendation. Because there is a lack in implementations and it is not a standard, yet I will not expand on any further details and I rather continue with another, more centralized approach toward service coordination.

2. Content-based routing

Content-based routing uses adoption of enterprise integration pattern called Content-based router (Source [2]). It is centralized approach taking advantage of having one routing service, which works as a router and all communication goes through this router. The service has routing rules defined in its logic. It is also stateful. The routing rules may be defined in a process modelling language like WS-BPEL (Web Services Business Process Expression Language) or Javacript.




**Picture 2-2: Content-based router**

Content-based routing is also called orchestration due to its parallelism to a real orchestra. In real life, a conductor leads whole orchestra similarly as on centralized router routes traffic in ESB. In this work, I prefer term orchestration to express content-based routing.

Orchestration in compare with choreography is easy to implement. A process language WS-BPEL is OASIS standard. A many ESBs exist having WS-BPEL implemented and only few solutions have adopted some form of choreography. WS-BPEL is a standard, whereas WS-CDL is a W3C recommendation and itinerary-based routing is proprietary.

### 2.3.4. Extended definition of ESB

In chapter 2.3, I introduce Gartner's definition of ESB. This definition is very general. Gartner introduces only the key areas that ESB concerns, although ESB is a tremendous topic. There is a lack of printed literature regarding ESB (source [1]), but fortunately, I can draw on plenty of internet sources, mainly web sites of firms, which develop ESB products like Sonic, Apache, Bea, Sun or Oracle.

ESB is considered to possess several characteristics. This set of capabilities of ESB varies from source to source. The list below enumerates common characteristics. Some of them are not widely agreed, but I decided to expose them with my comments.

1. Reliable communication using messages

   Services can communicate one another by sending and receiving messages in various possible modes: asynchronously, synchronously, point-to-point, publish-subscribe. Messages may be persistent; it means they may be stored persistently whenever service is not available. The technique that may be used to secure this behaviour is called durable subscription[2].

2. Mediation

   ESB is a layer between providers (producers) and consumers of services and therefore it supports their loose coupling without having to mutual realization of each other.

3. Independence on operation system and programming language

   It enables communication between application having made by using different programming language and running on different operation system.

4. Usage of XML as a message format

   XML brings many advantages like platform independence, wide support in various programming languages and applications, transparency; it is human readable, which is very useful for monitoring and administration of system. In spite of plenty advantages, there is a disadvantage, which cannot be taken lightly. Usage of XML might ends up with performance problems, because XML message is often larger than other proprietary or binary format. Overall XML format shows its advantages in Web Services.

5. Foundation of standards

   A very important task is to reduce proprietary technologies and prefer open standards. For example HTTP, JMS, J2EE Connector Architecture, JMX, SOAP, XML, XSLT, XPath, XQuery, WSDL, BPEL, WS-*, etc. Older integration architecture EAI used proprietary standards, because there was a lack of appropriate open standards then.

6. Orchestration services support

   Communication between services may be part of a process flow and it is controlled by orchestration service. It may be implemented as a service what joins to communication bus and controls message flow and calls among other services. Process flow may be long-term and stateful. EAI hubs implemented orchestration, but in much centralised way. Orchestration may not be centralised, process flow can be controlled by several interconnected orchestration services.

[2] More information about durable subscription you can find within [4]

7. Security model for authorization, authentication and audit service of communication bus

   Among ESB nodes there might be firewalls without having any affect on ESB functionality. Therefore, ESB may cover several network segments. It is even possible to have ESB spread globally. Thanks to usage of HTTP as a transport protocol, request can pass over firewall, however, there is a question, what kind of request may firewall consider harmless and let is pass. A turning point might be arrival of some kind of SOAP virus, which could lead to more restrictive firewall policy. More likely, WS-Security protocols will offer appropriate solution even for case above. In many cases, ESB security model may rely on MOM security model wherever it is possible, for example, in Open ESB.

8. Distributed and progressive deploy

   Applications can be attached to the bus progressively as well as needed infrastructure. It is a great advantage, in particular, because part of systems may run on legacy infrastructure and the rest may be already attached on ESB. Therefore, integration is not necessarily an invasive process. In contrast with EAI hub-and-spoke architecture, infrastructure can be highly distributed whereas EAI requires having the infrastructure centralised around hub(s).

## *2.4. Real-life case study*

### 2.4.1. Airport: real-life case study introduction

Real-life case study shall reflect real-life requirements. It would to be the proof that given framework is capable of providing quality implementation meeting all requirements. This case study is supposed to be as close to reality as possible.

Case study description:
An airport boss under the pressure of competitive environment had agreed to create an online ticket booking application. The airport uses services of three airlines: Czech Airlines, Lufthansa and British Airways, for instance. All three provide interfaces (through Web Services) to their internal booking mechanisms.

Task:
The task is to create an application having linked to every airline that fulfils search criteria, downloads results of search (lists of available flights for every airline), merges it together and possibly sorts, and eventually displays results to the end user.

Justification:
This is a real-life example; many internet portals work already this way. Of course, there could be even more use cases, for instance, a booking of hotel in flight destination or a creating of a payment services (like VISA and MasterCard) and involving them within application. However, these use cases are, basically, the same as fly ticket ordering in terms of creating links to several endpoints, downloading and merging list of results. Thus, I have decided not to involve them into solution.
As for technology to provide interfaces to these services, I have decided to use Web Services (WS), because every ESB framework regardless of its type supports them natively.

# 3. Elaboration

## *3.1. Picking frameworks*

This work elaborates three different approaches of system integration. First is ESB, the second one is BPM and the concept of a popular proprietary framework.

In time of elaboration of this work, it exists above 30 implementations of ESB. Some of them are commercial, whereas the rest is open-source. In terms of quality, open-source implementations are often as good as commercial. Their advantage is the price. They are free. Consequently, a potential customer may lack levers that could push the community of developers being involved into making an urgent fix for discovered defects, if any. Therefore, many open-source frameworks offer the option of getting a paid support. This condition is the most important one for enterprise clients. I decided to prefer open-source integration frameworks with possibility of getting paid support.

As I have written above, the support is often the most crucial thing for many projects. The quality of support goes hand-in-hand with popularity. If certain framework is more popular than another is, very likely it has wider foundation of developers working on it. Its development process therefore proceeds quicker. Furthermore, they tend to adopt new ideas and to come up with remedies swiftly. These are the reasons why I pick popularity as another condition. Regarding programming language, Java is the best choice for integration, because it is platform independent programming language. Thus, I decided to impose the condition allowing only java-based frameworks. This condition is meant to filter platform depended frameworks, for example, products of Microsoft. Fortunately, the vast majority of frameworks is java-based.

Condition summary:

- open-source framework with possibility of having paid support or training
- java-based framework
- popularity and adequate developers community
- representativeness

SOA and ESB do not exactly define all the implementation details. It causes that integration frameworks may be very different from one another. Some of them use BPEL as an orchestration language; next ones use some kind of modification of ESB patterns to improve integration itself. According to [1] it is even possible to use scripting language (javascript) to service orchestration (Apache ServiceMix). I decided to filter out implementation not supporting BPEL, because this execution language was tailored to orchestrate service.

Aforementioned criteria match with following ESB frameworks: Sun Open ESB, Apache ServiceMix, Apache Synapse, Celtix and Chainbuilder. Probably there are more albeit less known.

I have chosen Sun Open ESB framework, because it matches the imposed conditions most of all. Apache Service Mix is very similar to Sun Open ESB, but it does not provide graphical user interface. In addition to others, Open ESB has an advantage of support incorporated within NetBeans IDE.

On the field of BPM, there is the only suitable product that meets all the criteria. Its name is JBoss jBPM. As for the popular proprietary framework, the most suitable option has turned out to be Mule by MuleSource. Mule has a number of success stories. It has been deployed at Deutche Bank, CitiGroup, American Airlines or Adobe. Its programming model promises to be a respectable opponent of other two frameworks.

## *3.2. Framework Evaluation Criteria*

This chapter expands on evaluation criteria set to provide unified view on the frameworks. Their purpose of the criteria is not to be use as a metric of comparison in terms of comparing frameworks one another. In most cases, it is not even possible due to the vast difference in nature among them. These criteria help to discover design discrepancies, implementation problems, useful features, code generation possibilities and other capabilities. Some of the criteria may not be applicable for particular framework. In that case, they may be omitted. In the end, the information gained shall suffice to make a comparison of the selected concepts.

I focus on following criteria:

1. usage of standards and proprietary technologies
2. quality and coverage of available documentation
3. installation, configuration, integration with other apps/tools
4. code generation possibilities
5. graphical representation of business processes
6. availability and quality of IDE[3] plugins
7. technical knowledge required to work with given framework
8. helpful tools and features (to manage or speed up development process)
9. availability of connectors and adapters to connect to another system
10. scalability, clustering, load-balancing, failover
11. outlook and future plans

I expect that the comparison of chosen frameworks in terms of real-life case study will be helpful for business people as well as technical leaders or architects to decide, which framework will be the best choice for their project.

[3] IDE stands for Integrated Development Environment. For example, Eclipse or Netbeans

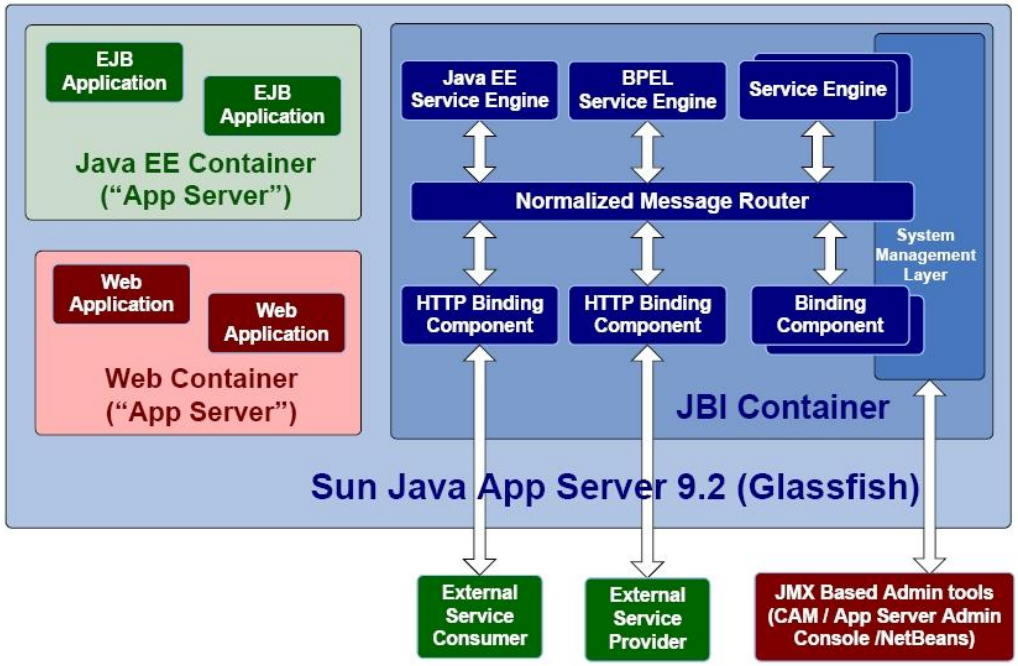
## 3.3. Sun Open ESB

### 3.3.1. Project background

Sun has started Open ESB project on May 2006 as an open-source alternative to its commercial ESB called JCAPS. It has been created from scratch on the very best foundation of JCAPS. Open ESB is made upon common ESB standards, which are kept very tightly. Project has developed rapidly having concerned not only bare ESB, but also remarkable toolkit tightly related to another Sun project NetBeans, which is powerful Java IDE. Open ESB has adopted JBI[4] standard enabling it to use JBI components from different sources, although Open ESB itself has large repository of JBI components.

### 3.3.2. Architecture

Picture 3-1 shows the architecture of Open ESB.




**Picture 3-1: Open ESB Architecture overview**

Source [6] describes Open ESB as a platform hosting a set of pluggable component containers, which integrate various types of IT assets. These pluggable component containers are interconnected with a fast, reliable, in-memory messaging bus called the Normalized Message Router (NMR) also referred to as the JBI Bus. Service containers adapt IT assets to a standard services model, based on XML message exchange using standardized message exchange patterns (MEP) based on abstract WSDL. This

[4] Specification for implementing SOA that introduces a pluggable architecture built upon web services.

improves interoperability and allows a mix-and-match of technologies from various vendors. When sending and receiving messages outside the JBI environment, the engine component containers communicate using the in-memory NMR messaging infrastructure and pass messages out to the client through an appropriate binding component container. When communication is entirely within the JBI environment, no protocol conversion, message serialization, or message normalization is necessary because all messages are already normalized and are in standard abstract WSDL format.

Nevertheless, Open ESB has an unpleasant limitation. It requires open source Java EE Application Server GlassFish as a runtime environment. In spite of that fact, ports to other environments have been released on Open ESB Wiki page (source [9]). For example, there are ports for JBoss AS, WebSphere AS and even for Java SE making Open ESB run stand-alone. However, some of them are not still in final version, still they are here as an option.

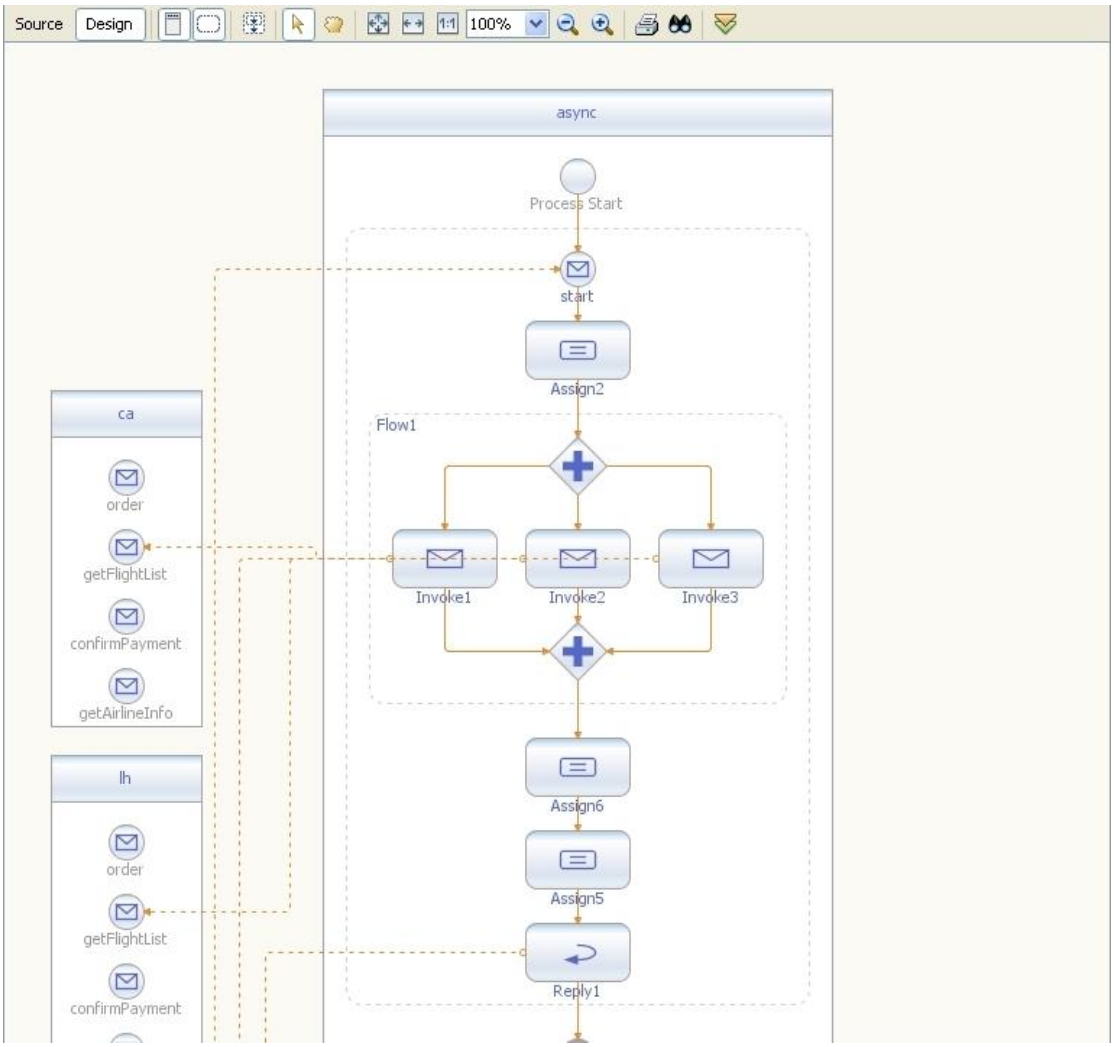
### 3.3.3. Available toolset

NetBeans 6.0 with SOA plugin contains several advanced tools for XML and BPEL. These tools are what makes NetBeans far more powerful IDE at SOA development than, for example, its greatest competitor: Eclipse. Let me take a closer look at them.

1. BPEL designer

   The intent of this unique open-source tool is to simplify work with BPEL as it is basically a xml document in specific format in design stage. A vast documentation is available on NetBeans project page [13]. The most important parts of BPEL designer are design view, palette of BPEL elements and BPEL mapper.

   The design view depicted at Picture 3-1 allows developer to add external services (known as partner links, located on the left side) as per previously prepared WSDL files. These services are going to be called from orchestration service (in our case it is BPEL service engine) through BPEL element “invoke”. Every BPEL process must be started by invoking (passing input parameters though SOAP message, for instance) element “start”. Element “assign” is used to assign values (numbers, strings, dates, etc.) to BPEL variables or to assign one’s variable value to another with possible modifications. The use of element “reply” allows developer to send a response containing results. Next abilities of BPEL mapper are discussed later in this paragraph. As you can see at Picture 3-2 in an upper left corner, there are two buttons: Source and Design. Source button leads to XML view showing resulting BPEL code. Refactoring is supported, it means whenever you make changes in design view, they will reflect in source view and vice versa.

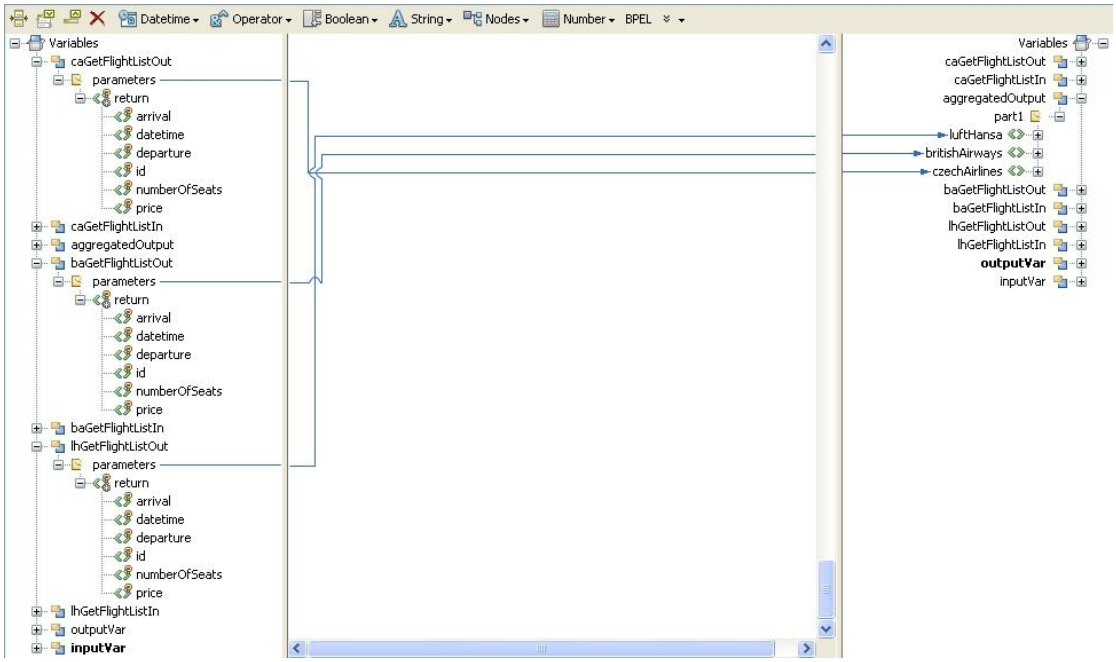



**Picture 3-2: BPEL designer: design view**

Palette of BPEL elements (Picture 3-2) is located in a panel on the right side of screen. It contains elements representing mostly XML elements of BPEL. These elements can be drag-and-drop moved to design view. Deeper description of particular elements is available on [13].


**Picture 3-3: BPEL designer: palette of elements**

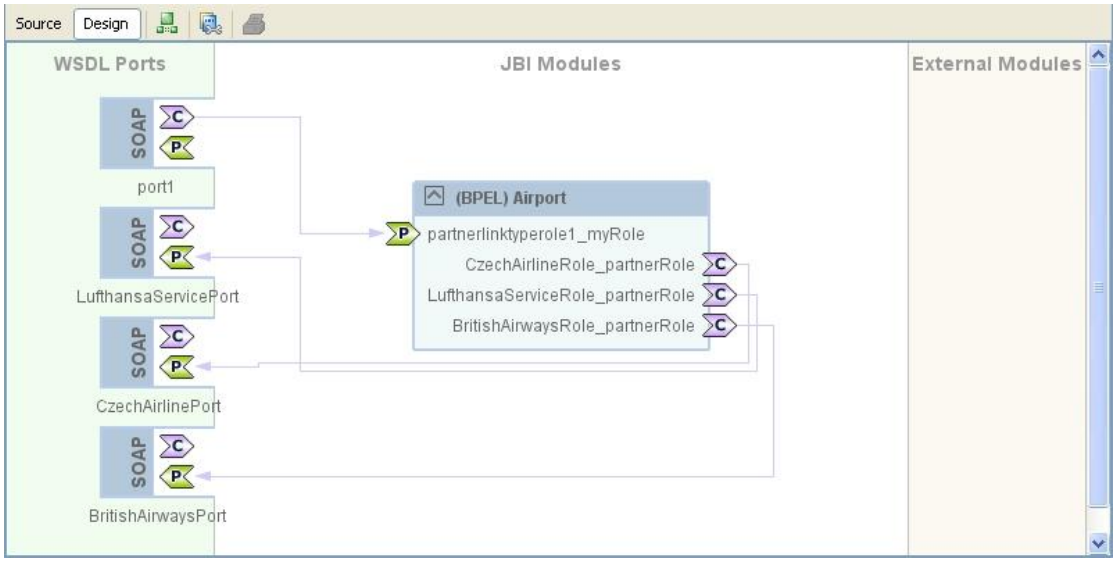
BPEL mapper is accessible through clicking on element "assign" in design view. In this view depicted at Picture 3-4Picture 3-4, BPEL mapper allows user to take advantage of assign element almost fully having all work with variables including copying, assigning values (either value of variable or a new value), converting types, using simple operations on strings (concat, suffix, prefix, etc.), numbers (basic mathematical operations), booleans, dates and times or to use XSLT transformations. The list of operations is still not exhausted. More information you can find at [13]. BPEL mapper simplifies also representation of xml elements and allows you to work directly with nodes and leafs and their content as is depicted on picture below. All assignments always apply from the left side to the right.




**Picture 3-4: BPEL designer: BPEL mapper**

2. CASA editor

The abbreviation CASA stands for Composite Application Service Assembly. It is a follow-up to a previous interpretation regarding BPEL designer. BPEL designer enables you to create an orchestration flow in BPEL, but it says nothing about related connectors or adapter, generally, it has nothing to do with how external systems are connected to ESB. To make that happen, you have to use CASA configuration editor.
The principal is simple. As you surely remember, during working with BPEL designer, you had to specify external services' interfaces, commonly using WSDL description language. Now, the point is to link these interfaces with specific binding components. Open ESB included in GlassFish Server contains a few embedded ones. For example, file binding component, HTTP binding component and JMS binding component. However, it is no problem to install other ones, from open-source binding component repository, for instance. This linkage is just creation connections among WSDL ports and JBI modules, as it is shown at Picture 3-5. These SOAP ports might be easily changed with JMS ports with no effort. Now we can see the main advantage of Open ESB and it is to separate configuration and designing flow.



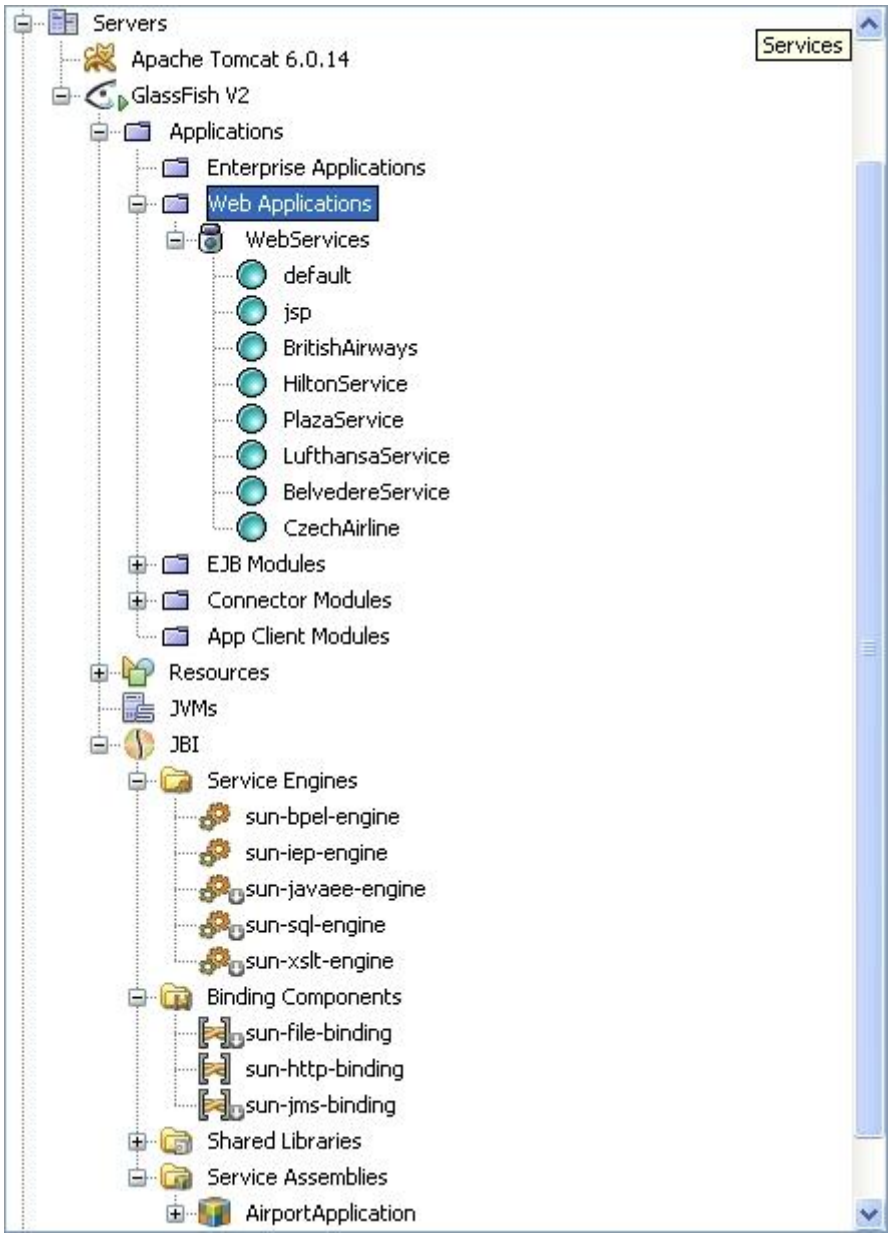

**Picture 3-5: CASA editor**

3. GlassFish Plugin for NetBeans IDE

Since NetBeans IDE is tightly linked to the application server GlassFish, it contains a useful plugin to handle GlassFish directly from development environment. Its user interface is depicted at Picture 3-6. The plugin for the GlassFish V1, V2, and V3 (Early Access) and Sun Application Server 8.1 and 8.2 (J2EE SDK Reference Implementation) and GlassFish for NetBeans is bundled with NetBeans 6.0, 6.0.1 and 6.1 Beta a free IDE that allows a developer:

- to create a Java EE application, automatically add EJB modules and Web modules, and deploy the application.
- to create an EJB module and deploy it as a stand-alone module or packaged in a Java EE application.

- to create Session Beans, Entity Beans and Message-Driven Beans.
- to create Entity Beans using an existing database schema.
- to create a Web module, generate calls to EJBs, and deploy the Web module either as a stand-alone Web application or packaged in a Java EE application.
- to create, register, and test Web services.
- To try out the sample applications to learn and understand the J2EE technology.
- To import your existing Java EE projects.
- To validate your applications using the J2EE Verifier.
- To visually configure your EJBs, Web Services and Web Components.
- To add multiple source folders to EJB module or Web module create Unit tests as a part of the project.




**Picture 3-6: GlassFish Plugin for NetBeans IDE**

At the Picture 3-6 you can recognise bundled Open ESB's service engines and binding components like BPEL engine, JAVA EE engine, etc. (warning: EIP engine is not default). A gray arrow next to particular engines/components indicates whether that engine/component is enabled or not.

### 3.3.4. Installation

Installation of Open ESB is very easy, if you use GlassFish (or its commercial version) and NetBeans IDE. Everything you need to do is just to download NetBeans "All" bundle with embedded GlassFish and SOA plugin. An installation program will do everything for you. There is a need for manual installation if you are going to use other application server than GlassFish or other IDE than NetBeans. Then you must follow the manual installation guide for Open ESB located its web page.

### 3.3.5. BPEL Service Engine configuration

BPEL SE is the most important part of Open ESB. Thus, it is crucial to have it configured properly, particularly in production environment.

Clustering is the way to solve a needed massive throughput of application, when application is deployed on several servers and the BPEL Service Engine's clustering algorithm automatically distributes processing across multiple engines. To configure clustering, deploy your composite applications across multiple BPEL Service Engines running on multiple processors or systems.
When your business process is configured for clustering, the BPEL Service Engine's failover capabilities ensure throughput of running business process instances. When business process instances encounter an engine failure, all available BPEL Service Engines pick up any suspended instances.

There are several conditions you have to meet prior to start of clustered environment. Firstly, deployment has to be done on every machine manually; these machines have to be in the same time zone. Every BPEL SE must run upon the same single database and its high availability is crucial, because when DB fails, clustering will fail. It supports JavaDB, a Sun's supported version of an open source Apache Derby database, by default. However, it supports Oracle DB as well. Finally, you have to avoid inbound message activities (Receive, On-Message, On-Event) within BPEL flows in order to secure failover.

### 3.3.6. Support

Sun Microsystems has created two ESB solutions. One of them is Open ESB, which is free and another product is commercial Sun Java Composite Application Platform Suite (Java CAPS or JCAPS. It is available only for authorized JCAPS partners. The relationship between Open ESB and JCAPS is complicated.
Paul Perez, co-founder and Chief Software Architect of Pymma (consulting company, technical architects office, expert in the design and implementation of high performance Java EE and Java Business Integration based architectures with high-capacity of integration) explains [7] the Sun's commercial product policy as follows. Sun offers two ESB products: JCAPS is a commercial product inherited from Seebeyond, whereas Open ESB is based on JBI specifications (JSR 208) and was developed from scratch about 2 years ago. These products are not following the usual Sun's policy. While other Sun's product like Sun QM and its open source version Open-QM shares approximately 95% of code and the only difference is commercial support, JCAPS and Open ESB are totally different product at the moment. Let me list a several differences:

1. JCAPS ignores JBI specifications
2. JCAPS connectors are based on JCA specifications and not on JBI
3. Open-ESB development process is based on Web services specifications, JCAPS not
4. JCAPS and Open-ESB developments are not compatible

The question is why Sun would support two different products doing the same thing.
Many companies believe in JBI and their developers spend time and energy working on Open-ESB. These companies would certainly be interested to hear Sun's explanations on the above questions. They probably want to be sure that Open-ESB will not be just a prototype for the new JCAP version (only reserved for SUN JCAPS Partners). They certainly want to be credible by proposing Sun's professional support on Open-ESB as they do for Glassfish and Open-Solaris.
JCAPS is, at the time of this work, in Release 5. JCAPS Release 6 is just around the corner (Q2). In JCAPS Release 6, Sun will be moving towards the same model as it does for all SW products. That is, open source development with commercially supported builds.

The JCAPS story is a bit more complicated because of backwards compatibility issues. That is, Release 6 will combine Open ESB technology in addition to Release 5 technology, which will remain in the product for backwards compatibility issues. However, the long-term plan is that eventually JCAPS will solely be a commercially supported version of open source technology. That is, Open ESB and related projects, for instance, Mural.
Sun officially does not support Open ESB. However, there are external companies, which do. One of them is aforementioned consulting company Pymma. In the future, with coming of new release of JCAPS, the support will be provided by Sun.

### 3.3.7. Case study implementation

This paragraph shows how to implement a given real-life case study in environment of NetBeans 6.0.
Assuming that airlines provide interfaces to their systems through Web Services I consider the easiest way, how to implement a real-life case study Airport (paragraph 2.4.1) as follows:

1. Deploy Web Services of particular airlines on GlassFish. Make sure that GlassFish is up and running.

2. Create a sample project: Synchronous (possible asynchronous) BPEL project. It will create a sample BPEL diagram for you. It is easier then to start with a diagram skeleton.

3. Import external WSDL document into Process Files in BPEL project you had created. (right click on directory Process Files -> new -> External WSDL document(s). Select import from online location. For example, use URL like http://localhost:8080/WebServices/BritishAirwaysService?wsdl[5].

4. In directory Process Files, create new BPEL process (right click -> new -> BPEL Process and open it.

5. Drag-and-drop WSDL files you had imported into a BPEL diagram you have opened. You will be asked to create partner links (links between external applications and BPEL orchestration service).

6. Create a BPEL diagram with BPEL designer described on page 23. Keep in mind that diagram must comply with WS-BPEL 2.0 (source [14]). It is not in scope of this work to explain in detail how to work with BPEL designing tool.

7. Create a project Composite Application. It is located in "New project" dialog in folder called SOA. This project manages configurations of our case study implementation. We can define what type of connection to external services will be used or generate a sample tests to test application, etc ... Build this project and thus create a jar.

8. Right click on our Composite Application project-> Add JBI module. Then import work from previous project. Then build this whole project.

9. Right click on our composite application project in projects tab, select "Edit Application Configuration". This opens you a CASA editor described on page 26. Here you can map you service description in WSDL to binding components (SOAP by default). In the middle section (JBI modules) of the CASA diagram, you may see BPEL module.

10. Build and deploy on GlassFish composite application you have created.

[5] This URL is more likely to be different in your case.

11. Optional step is to create test to check whether our application works fine. Right click on directory Test in composite application. -> Create Test Case. Follow offered steps, fill out test name and select WSDL document and operation, which you want to create a test to. It will generate a SOAP envelope with parameters, which you can modify. Then, if your application is up and running, you can run your test and check the results.

### 3.3.8. Case study implementation comments

Designing an application that uses flights search engines of three airlines is not difficult in environment of Open ESB. One of the difficulties I had to face is the fact that BPEL 2.0 does not solve aggregation of results very well. In this case, a very useful would be to put support for aggregation pattern (source [2]) into BPEL language. In concrete, support for time/count-based aggregation (saving messages and after certain time/count aggregate) is desirable. As and alternative to BPEL service engine in terms of aggregation issue, I would suggest IEP (intelligent event processing) engine which contains both kinds of aggregators by default. However, it differs in principles and therefore there is no need to go deeper. This problem is general; it means it occurs on other orchestration engines and ESBs. One non-traditional proprietary solution brings up Oracle (source [15]). It has extended BPEL language with additional elements in order to make it stronger to cover actual problem.
I decided to solve this problem using XSLT transformation. I created a variable in BPEL, which contains three XML SCHEMA sequences (type of particular airline). These sequences are holders for flight search results. Then I performed search and put the results into this variable (into appropriate sequences). As a last step, I used XSLT transformation to transform this variable into desired format, it means, to one long sequence. Nevertheless, this approach is not desirable, very appreciable would be means like special aggregators provided by BPEL standard itself.

### 3.3.9. Framework evaluation

Let me evaluate systematically all criteria imposed in one of the previous chapters na stránce 21.

1. usage of standards and proprietary technologies

   Open ESB supports crucial ESB standards, namely OASIS Standard WS-BPEL 2.0 (BPEL) along with WSDL 1.1 and JBI 1.0. A complete list of constructions supported in BPEL Service Engine is available [11]. Evidently, the most important constructions (like invoke, variables or receive/reply) are supported almost fully, other functionalities has only limited support. In spite of that, BPEL support in Open ESB is sufficient. JBI 1.0 and WSDL 1.1 are fully supported.
   Thanks to JBI specification, which Open ESB is convenient to, this framework is easily extensible with additional binding component and service engines either from another JBI convenient frameworks or from independent external JBI component repository (for example [10]).

2. quality and coverage of available documentation

   Another part of problem surprisingly appears to be documentation on official Open ESB page [6] and Wiki [9]. At the first sight, it seems quite complete, full of screenshots and examples. On the other hand, there is no exhausting documentation describing Open ESB continuously from the beginning. Examples are useful, but they are useless, when you really need to solve uncommon, specific problem. Nevertheless, behind Open ESB is large social community, many contributors have their own blogs, where they discuss problems that appear during implementation or testing. To sum up, realising that Open ESB is started two years ago, quality of documentation matches with its age. With support from Sun, there is a good outlook for documentation to become larger with broader mesh.

3. installation, configuration, integration with other apps/tools

   Installation of Open ESB along with NetBeans IDE and GlassFish AS is easy. Open ESB has a solid support of clustering and failover, two the most desired features in production environment. Both installation and configuration are explained in detail in paragraph 3.3.4 and 3.3.5.

4. code generation possibilities

   Code generation, in its nature, is not a feature of Open ESB. However, code designers, which are part of NetBeans IDE, namely BPEL designer, WSDL and XSLT designers, can rapidly speed up whole development process and reduce a necessity of deep knowledge of aforementioned standards. On the other hand, developers have to spend some time to learn to work with designers. Code generation is noticeable when you creating tests for BPEL flow. It allows you to have whole SOAP envelope (input) generated; your only task is to fulfil input parameters. Envelope structure is generated from WSDL description. Overall, this makes testing easier.

5. graphical representation of business processes

   BPEL designer provides quite illustrative view on business processes. In reality, it depends on how the person, who is responsible for creating business processes in BPEL, is capable of depicting those processes clearly and briefly to make it as readable as possible for business people. However, BPEL is still an execution language full of details related to implementation.

6. availability and quality of IDE plugins

   Contrary to previous Open ESB limitation to GlassFish, this link between Open ESB a GlassFish has brought also many advantages in terms of extended toolset for NetBeans IDE or for example Web and EJB container, which are often use along with ESB technology. NetBeans environment seems to be the best option for developing Open ESB solution, even thought the leader among IDEs is its competitor, Eclipse. Support for Eclipse is limited, whereas NetBeans contains several useful plugins to facilitate work with GlassFish as well as with Open ESB. The most important are BPEL designer and BPEL debugger. Very useful is CASA editor, handy configuration editor, which puts BPEL projects and their configurations into deployable JBI assembly. That is still not enough, NetBeans offers powerful editors for XSLT, XML SCHEMA and WSDL. On the other hand, these editors are available in Eclipse, as well. Although NetBeans tools are surely useful, they still have their bugs and gaps. For instance, tooling for XSLT transformation in BPEL designer is still not complete, you have to input specific string in order to locate XSL sheet (like urn:stylesheets:aggregate.xsl). I hope that these gaps will be soon removed.

7. technical knowledge required to work with given framework

   Assuming that developers have already at least intermediate knowledge of XML, Java and XSL(T), then they are required to learn WSDL and work with BPEL designer and, of coarse, the principals of ESB along with BPEL. CASA editor (editor of configurations) is also important tool to learn, although work with it is easy. Available documentation provides many examples of work with both designers, although during development a specific kind of problem might appear. Then a deeper knowledge of BPEL standard (or other) is necessary, as it happened during this case study implementation. It is hard to estimate, it depends from person to person, how fast is somebody capable of learning. Nevertheless, the list of technologies and standards is not so large and many examples are published on project's web page and wiki. That makes Open ESB easy to learn, especially, if you experienced person in your team.

8. helpful tools and features

   Open ESB have several tools and plugins for NetBeans, which facilitate and speed up whole development process. For more information, see page 23.

9. availability of connectors and adapters

   Thanks to support of JBI, Open ESB has a solid foundation of service engines (BPEL SE, Encoding SE, Enterprise Data Mashup SE, ETL SE, Intelligent Event Processor SE, Java EE SE, SQL SE, XSLT SE ...) and binding components (CORBA BC, EJB BC, EmailBC, SMTP BC, HTTP BC, FTP BC, File BC, JDBC BC, JMS BC, MQ Series BC, MSMQ MC, SAP BC, TCPIP BC, etc.). There are several SE and BC planned.

10. scalability, clustering, load-balancing, failover

    Open ESB supports clustering and failover. For more information, look at the paragraph 3.3.5.

11. outlook and future plans

    The future of Open ESB is bright. Many binding components and service engines are being tested and going to be released in next few months, plenty of them are planned. A support from Sun is going to be consolidated. Open ESB version 3 has been announced. Version 3 combines strong modularity support of OSGi[6] with JBI infrastructure of Open ESB, to create robust, agile open source service-oriented integration platform. he core framework for Open ESB v3 is implemented at Project Fuji. Check it out at http://fuji.dev.java.net. With Fuji at its core, Open ESB will run on any OSGi R4 compliant runtime, providing a tremendous amount of flexibility in your choice of deployment environments, from Java SE, to GlassFish, to other OSGi-based application servers.

## *3.4. JBoss jBPM*

### 3.4.1. Project background

JBoss jBPM is a JBoss Enterprise Framework that delivers workflow, business process management (BPM), and process orchestration in a scalable and flexible product footprint. (taken from official jBPM datasheet [16]).
It started as an open-source workflow engine in 2003 under the Lesser General Public License (LGPL) and in October 2004 jBPM joined JBoss to become a critical piece of the JBoss Enterprise Middleware Platform. Since it is an open-source project, you may find project-related files as well as documentation at SourceForge.com.
Since jBPM is a mature project, it has many success stories, namely it has been used by British Telecom, SNS Bank, Alfresco, CadTel and plenty others.

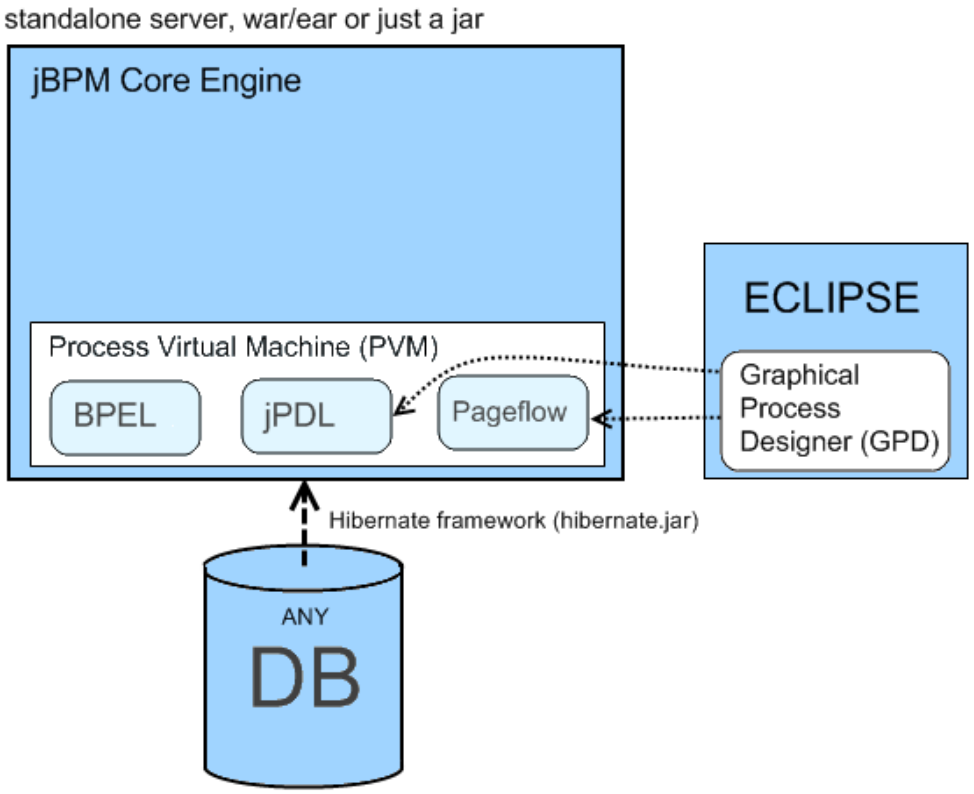
### 3.4.2. Architecture

Open ESB has provided a flow-control based on BPEL, which is a business process execution language. It provides low-level view on business processes. Therefore, the development process is performed by developers along with business and continues from below up. BPEL cares about integration from low-level point of view. In other words, it cares only about how to instruct certain services to create one or more flows representing business process. It works with services, not business units.

[6] OSGi – a dynamic module system for Java

To the contrary, BPM (business process management) uses from above down approach. Firstly, business process modeller creates a model of business processes in organization unit. Every process must be mapped and described. This approach can possibly lead to optimization of particular processes. This makes it easier for non-developers (ordinarily people from business) to understand processes and making changes if necessary. Further, model is passed to developers to complete technical background like creating link between processes and services upon various technologies and making them collaborate. Developers gain brief model of processes, which is easy to understand. This approach can speed up whole development process rapidly. Even after successful implementation, businessperson may perform certain set of changes on high-level model making on-the-fly changes into production.

jBPM consists of three the most important parts: Process Virtual Machine, jBPM Core and GPD (Picture 3-7: jBPM architecture**Chyba! Nenalezen zdroj odkazů.**). There are also parts like Hibernate[7] libraries, which secure database connectivity to any database. Its role is optional, but not avoidable if you want to use long-term transaction or failover. Then all states are stored to database.




**Picture 3-7: jBPM architecture**

[7] Hibernate is an ORM (object-relational mapping) framework. Basically, it maps database structures to objects in memory and thus it creates data-access layer.

JBoss jBPM features in detail:

1. Process Virtual Machine

Native support for any process language can be build on top of the Process Virtual Machine. The runtime behaviour of each activity in the process graph is delegated to a Java interface. Process languages are a set of activity types. An activity implements the runtime behaviour and corresponds to one activity type. So building a process language on the PVM is as easy as creating a set of activity implementations. Through the same mechanism, languages like JPDL are very easily extensible.
JBoss jBPM supports through PVM three languages namely BPEL, de facto orchestration standard.
Next supported language is Seam Pageflow. jBPM uses it in integration with JBoss Seam framework. Seam offers the possibility to use jBPM Pageflow as the navigation mechanism between web pages. Creating these pageflows graphically is supported by the GPD (graphical process designer).
What leads us to the last supported language JPDL that is jBPM's preferred process language. While directly executable, it provides excellent modelling capabilities. It leverages the PVM and the Identity, Task Management and Enterprise components and thus provides excellent support in these areas. Graphical creation of JPDL process definitions is supported by the GPD.

As JPDL is the most preferred language supported by jBPM, it is going to be used for evaluation of our real-life case study, whereas Seam Workflow has not be designed to business integration and BPEL would not lead to discovery of something valuable anymore.

2. Graph-oriented programming

There are numerous graph based process languages. There are big differences in the environment and focus. For instance, BPEL is intended as an XML based service orchestration component on top of an Enterprise Service Bus (ESB) architecture. In addition, a Pageflow process language might define how the pages of a web application can be navigated. These are two completely different environments.
Despite all these differences, there are two features that you will find in almost every process language: support for wait states and a graphical representation. This is no coincidence because it's exactly those two features that are not sufficiently supported in plain Object Oriented (OO) programming languages like Java.
Graph Oriented Programming is a technique to implement these two features in an OO programming language. The dependency of Graph Oriented Programming on OO programming implies that all concrete process languages, implemented on top of Graph Oriented Programming, will have to be developed in OOP. However, this does not mean that the process languages themselves expose any of this OOP nature. E.g., BPEL is not related to OO programming and it can be implemented on top of Graph Oriented Programming. It is not in intention of this work to explain JPDL or graph-oriented programming in detail. More information you can find in jBPM official user guide.

3. jBPM Web Console

The jBPM web console web application serves two purposes. First, it serves as a central user interface for interacting with runtime tasks generated by the process executions. Secondly, an administration and monitoring console allow inspecting and manipulating runtime instances. The third functionality is Business Activity Monitoring. These are statistics about process executions. This is useful information for managers to find bottlenecks or other kinds of optimisations. jBPM web console is packaged as a web archive (war) and can be easily deployed to any server.

4. Long-term processes

Thanks to Hibernate, jBPM is ready to use persistence as way of solving failover. Then every state of process is stored into a database. If any problems would occur, whole process and its process state can be restored without a data loss and it might carry on. Second part, necessary for long-term processes is support of wait states. Both of these conditions were met with jBPM.

5. Human interaction

If a company has a difficult process diagram with many states and exceptions out of the flow, it might happen that, for example, one state is blocked (external application is down, etc. ). jBPM offers a possibility for administrator to push blocked process to continue working. jBPM also provide human tasks management. It is a way to interact with flow by inputting parameters or results manually by a person who has appropriate rights. For instance, certain tasks have to be approved by the highest manager. Therefore, there is no need to develop extra application securing data input, jBPM contains powerful web console. It allows users, managers or administrators to see instances of processes, their current state and many more. In addition, when process enters human task state, user is invited to fill out offered form. jBPM can generate a web form for its web console out of input parameters. This ability is very handy.

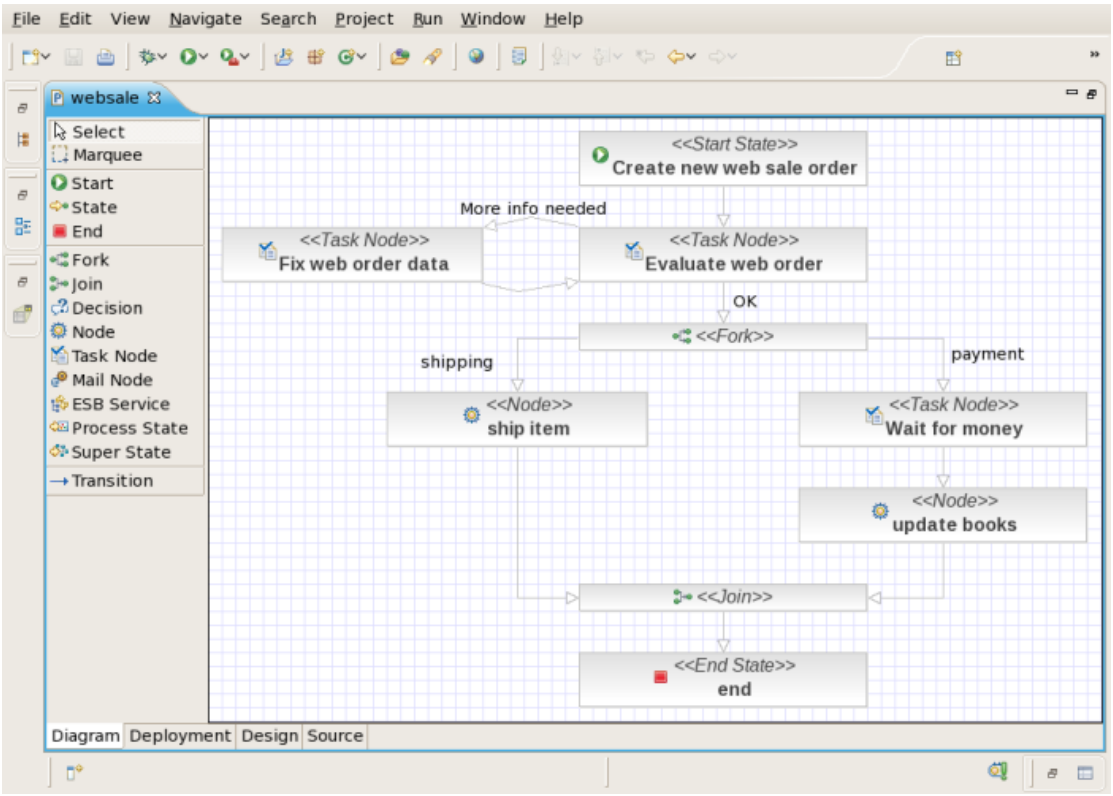
### 3.4.3. Graphical Process Designer (GPD)

JBoss jBPM becomes interesting evidently for an ability to create a process diagram with a graphical tool. This diagram is then understandable for business people as well as for anybody else, like developers. This ability is available thanks to aforementioned GPD. It is an Eclipse plugin (unfortunately with no support for NetBeans or other frameworks) consisting of several views. GPD supports two languages, JPDL and Seam pageflow. When Seam started to use jBPM for page navigation, support for Seam pageflow was added. Only JPDL is going to be in our focus from now on.

The purpose of GPD along with JPDL is to create a process diagram that describes business processes in detail preferably with all the exceptions in flow (time out, unexpected situation) to cover every possibility of flow. The output is process description in XML based language, JPDL.

Picture 3-8 is an example of process diagram. Right side represents a workspace, where user drag-and-drop elements from a JPDL elements repository on the left side. As you can see, many of those elements represent certain states in process diagram, whether it is start, end, fork or join. The depicted diagram shows a book ordering process. If one diagram is not large enough to describe all states of the process briefly, it may be divided into two or more diagrams describing sub-processes.

GDP provides two options to complete process description. One is to create a process diagram, another one is to switch to Source view (in the bottom of the picture, there is a tabbed view) and fill out JPDL description manually. JPDL is a language based on XML. Even though in most cases you would never have to work with JPDL code directly, GDP does not cover whole JPDL and therefore, under certain circumstances, you would find this possibility very useful.




**Picture 3-8: GPD example**

### 3.4.4. JBoss jBPM basics

JBoss jBPM is in close relationship with process-oriented programming, graph-oriented programming as well as the term known as a workflow. In fact, it is built upon them. Therefore, an understanding of these concepts is vital for moving on.

Firstly, we have to examine what the process really is. There is no widely accepted definition, despite the discussion having been hold since 1990s. One of the most prominent definition was given by Hammer and Champy (source [17]), who were evaluating external behaviour of processes.

They claimed that process is

*"a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer".*

In a more refined approach, Davenport proposes the following definition:

*"A process is a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action"*

A proponent of Total Quality Management efforts, Harrington, defines a business process in the context of his Business Process Improvement approach as:

*"any activity or group of activities that takes an input, adds value to it, and provides an output to an internal or external customer. Processes use an organization's resources to provide definitive results."*

More specifically, he distinguishes between this (general) process and more specific production and business process. In a similar definition, Schmidt describes a business process as:

*"stepwise procedure for transforming some given input into some desired output. The transformation is consuming or using resources. A business process has some form of outcome, i.e. goods or services produced for a customer or customers either outside or inside the enterprise."*

The above definitions are similar with regard to the dynamic system view employed; they all stress the input-process-output character of organizational processes. Nevertheless, these definitions are not precise enough to allow for a detailed description of organizational processes using information models. For instance, the definitions do not address the criteria separating different processes within an organization, which is a crucial distinction for the development of organizational process models. In the context of this work, we define a process in the tradition of Becker and Schütte as follows:

A process is a discrete, holistic, temporal and logical sequence of those activities that are necessary to manipulate an economically relevant object. This object is also called the process object and characterizes the process. Additional supporting objects may become part of the process. Depending on the properties of the process object, we can distinguish between material and information objects.

Business processes are a specific category of processes. A business process is defined as a high-level process determined by the overall goals of the enterprise. Business processes contain activities that interface with market partners (i. e., customers, suppliers, or other third parties).

A workflow is a specific representation of a process, which is designed in such a way that the formal coordination mechanisms between activities, applications, and process

participants can be controlled by an information system, the so-called workflow management system.

JBoss jBPM recommends using JPDL (jBPM Process Definition Language) along with GPD. Then particular process may be represented as directed graph with at least start and end element, where nodes are JPDL elements representing states (fork, join, task, decision, etc.) and lines are transitions between states. Every process starts in Start Node then it is simply evaluated according to states and transitions and it ends in one of the End Nodes. This representation is clear and readable for anybody. Previous refers to graph-oriented programming, what (along with GPD) is explained on page 38.

Process is a definition. It may be instantiated creating one or more instances of process. Instance contains current state(s), process definition and instance variables. It is always possible to determine, which the current state is.

The most important JPDL nodes types are:

- start-state: starting node, always single
- end-state: ending node, one or many
- task-node: contains one or more tasks, might be used for human interaction
- decision: automatic decision (splitting the flow) (value > 1000 then ...)
- fork: creates two independent branches providing parallel processing
- join: waits for all branches created by fork to finish

You may attach actions to these JPDL elements. JPDL offers you three possibilities to attach. During transition, before execution of the node or after execution of the node. An action is typically a java code placed in method called execute in a class implementing ActionHandler interface. Here is an example:

```
public class MyActionHandler implements ActionHandler {
  public void execute(ExecutionContext executionContext) {
        ... CUSTOM IMPLEMENTATION ...
  }
}
```

The input parameter of type ExecutionContext carries all necessary information about the process state and the state of current instance. With this context you, for example, may push the instance to another state.

### 3.4.5. Installation, configuration and deployment

JBoss jBPM is available at SourceForge.net as it is open-source project. There are a few download possibilities. The best way is to download JPDL suite consisting of jBPM server along with web console and Graphical process designer, although both may be downloaded separately.
Available is a jBPM with emphasis on BPEL, which contains a number of examples in BPEL. The site also offers a process virtual machine to download, what is a library for building executable state machines. It can serve as the foundation for any form of BPM, workflow and orchestration. All products are zip archives and contain product itself along with documentation, source files and examples of usage.

Installation of all product mentioned above is just a simple unzip of archive. The only product that requires an installation is Graphical process designer. It is installed as any other plugin to Eclipse.

In most cases, no configuration is needed, you may run jBPM server directly and deploy your creations to it. However, configuration may vary depending on input requirements. For example, if you want to use DB persistence, which is common need in production, you need to configure hibernate through its configuration xml file (hibernate.cfg.xml). Its configuration is well described in jBPM configuration and thus there is no reason to explain it in detail. To sum up, configuration of jBPM regardless of which parts of bundle will be used, is easy and smooth.

There are three possibilities, how to make jBPM work with your application.

1. use jBPM server, which is based on JBoss AS. Create a model in Graphical process designer and through it upload our .par archive along with all hierarchy of classes you had created.
2. incorporate jbpm.jar into your application

Deployment of jBPM web console, mentioned on page 38, is easy and depends on whether you use jBPM server of incorporation of jbpm.jar. The jBPM console is available as a war file, which is directly deployable web application and it works with every java web container. If you use jBPM server, you do not have to care about it, it is already deployed, and otherwise you must deploy it manually on server you will choose.

Both of these approaches have cons and pros. If you do not mind using jBPM server, which is modified JBoss AS, first option is the right one. Otherwise, the second option meets your expectation, however, you must expect a little bit more effort to configure jBPM, because probably you want to use a jBPM web console and you must manage process definition in terms of uploading, versioning and deploying.

### 3.4.6. Support

JBoss itself provides support for jBPM in terms of consulting, training and defect fixing. That is not all. JBoss has a network of partners (JBoss authorized partners) providing these kinds of services too. One of them is, for example, SoftPro. A person, who goes through completely training process and pass a final exam, will attain a certificate as a proof of his skills.

JBoss jBPM is an open source. It has a foundation of developers, which provides updates, changes and fixing on regular base. JBoss also pays a small group of developers and architects, who revise updates and create concept and future plans. JBoss carries out this practice on every project it covers. This politic has turned out efficient and low-cost.

### 3.4.7. Case study implementation

Case study implementation concisely consists of several steps:

1. create state diagram (process definition) in Graphical process designer
2. create actions for certain states of process
3. deploy of process definition
4. test and debugging step-by-step using jBPM web console

Whole concept is simple and basics are easy to understand. jBPM server is distributed along with examples, which can facilitate learning a lot. Documentation on official sites is sufficient and well structured. Good approach has turn out to be looking for jBPM best practices on the internet. People, who have already experienced common problems, give advices how to avoid common mistakes and thus quicken development.

Graphical process designer is a great asset. Despite a number of cosmetic defects in layout, it has met expectation. Web console has come out interesting and irreplaceable tool for debugging process instances and watching them.
I would like to point at the fact, that tools are sufficient and provide great performance, but what is more important is practice and experience.

It is not intension of this work to describe in detail coding of real-life case study, so I would rather move to evaluation of implementation.

### 3.4.8. Framework evaluation

Real-life case study has shown mainly positive things. JBoss jBPM has proven that it is mature project with good foundation, perspective and good support. Let us evaluate systematic bullets defined on page 19.

1. usage of standards and proprietary technologies

JBoss jBPM has brought a proprietary BPM language: JPDL, however, it allows to use standardized language BPEL. On the other hand, JPDL is no obstacle on the way when using Graphical process designer. It is still not perfect, but it does its work. In most cases, there is no need to learn all features of JPDL in detail thanks to designer.

2. quality and coverage of available documentation

Framework jBPM has documentation of higher quality. It does not only consist of a couple of examples, but it describes API and all necessary interfaces. Documentation to Graphical process designer is full of screenshot. On the internet, you can find a number of educational videos that instruct you in how to use Graphical process designer. To sum up, official documentation is good and sufficient. If you need more resources, internet offers several jBPM discussions, where you can always find answer to your problem.

3. installation, configuration and integration with other applications and tools

The first two have been described in detail on page 42. As a result, it is easy and with no problems. Framework jBPM offers interesting integration possibilities. One of them is integration with framework JBoss ESB, other JBoss product. JBoss ESB integrates jBPM to take advantage of business process management. Orchestration itself is then carried out by jBPM and JBoss ESB uses its JBI binding components as gateways to external systems. This integration is very interesting; more information is available in JBoss jBPM integration guide (source [18]). jBPM may be integrated into Mule framework. Mule itself supports orchestration only along with external orchestration engine. It provides BPM connector through which jBPM may be use to orchestrate services.

4. code generation possibilities

Graphical process designer can help the developer spare a lot of lines on code. Designing pure JPDL code, which is basically a XML file with specified structure slows down project as well as it removes many advantages of having business process diagram that is easy to understand even for non technical people. So Graphical process designer generates a JPDL code from a diagram and it allows to make updates both ways. However, it is the only code generation possibility that jBPM has.

5. graphical representation of business processes

Graphical process designer, which is described in detail on page 38, provides that kind of functionality. It does not cover everything that jBPM allows, but the coverage is sufficient. Despite of small defect in layout it is very useful tool. Web Console is the second way to work with graphical form of processes. It allows you to see process instances as a process diagram, where current state or states are highlighted. An

administrator is allowed to push the process instance from one state to another if, for some reason, it is stuck in one state. Web console is described in detail on page 38.

6. availability and quality of IDE plugins for given framework

JBoss jBPM is unfortunately available only for Eclipse IDE. However, it is possible to work with it on NetBeans, all advantages of graph oriented programming are gone, because graphical process designer works only with Eclipse. Despite a great progress of NetBeans, there is no information about preparing a NetBeans version of GPD.

7. technical knowledge required to work with given framework.

Besides the knowledge of jBPM framework, at least basic knowledge of Java and XML is required. Overall, the requirements for developers are not high.

8. helpful tools and features

The most helpful tools is unquestionably the GPD. It has been described at the page 38. There are no other tools know the author by this time.

9. availability of connectors and adapters

JBoss jBPM possess no real connectors or adapters. The user has to write its own ones. Other option is to take advantage of using the connectors of other framework. jBPM is just an orchestration engine.

10. scalability, clustering, load-balancing, failover

jBPM is a state machine: process descriptions and runtime status are persisted to a database but in a cluster they are not automatically failed-over. If a cluster node fails while executing some external trigger (UI, timer, JMS, etc) execution will stop and have to be started again. Depending on the transaction context you are in this could be done automatically (redelivered JMS, re-executed timer) or require UI interaction (error message for user if cluster node goes down requesting repeat).
Therefore, while the process description is persistent, workflow failover must be performed manually. jBPM can be used to build a fully failsafe, clusterable hot failover workflow solution, but it does not come out of the box.
For most cases, the best solution would be to encapsulate the jBPM API calls that you need in your application in a stateless session bean and cluster this last one. (Source [32])
Along with supporting business process flows embedded in Java applications, JBoss jBPM can be deployed with JBoss Enterprise Application Platform providing the clustering, caching, fail-over, load balancing, and distributed deployment features expected in a best-of-breed platform.

# 11. outlook and future plans

JBoss jBPM has become a part of the JBoss Enterprise Application Platform as strong framework for enterprise integration. From this point of view, its next development is secured by the involvement of the JBoss, so there are no doubts that this project should become obsolete in the close future. Since the designers do not reveal their plans in detail it is hard to say, whether the new orchestration language will complement the current trio. One thing is certain. The GPD will always keep up with new releases of Eclipse.

## *3.5. Mule*

### 3.5.1. Project background

Mule, concisely, is open-source integration solution based on Mule programming model with wide community behind and steady commercial support by MuleSource. It is licensed under the Common Public Attribution License Version 1.0 (CPAL).

The project has been started in 2003, which makes it one of the oldest projects on the field of enterprise integration. By the time of its founding, only few standards have been already released and put into common usage. For example, having focused on standard BPEL, its older version BPEL4WS concretely, was released in 2004, what was one year after project Mule has started.

Due to the lack of suitable standards, Mule has chosen a way of its proprietary architecture based on its own programming model. Its solutions were functional and it has gained many devotees. From the list of successful implementers, I can bring up CitiGroup, JP Morgan and Deutsche Bank as well as many other high profile enterprises including American Airlines, Adobe and Yum Brands, despites the fact, that most of the big players, prefer commercial solution to open-source.

After a several standards for enterprise integration were released and era of ESB based on BPEL has emerged, Mule gained unwanted competitors. To secure backward compatibility and to concern newly emerged standards, creators of Mule decided to implement both of approaches at once. That was the reason, why Mule has concerned, at least partly, concept of JBI, BPEL, BPM, etc. Therefore, it provides possibility for architect to pick the most convenient integration approach or to combine several kinds.

Nevertheless, the way of integration that Mule uses, has not been chosen unwisely. Its authors have adopted many of the integration patterns from the book Enterprise Integration Patterns (source [2]). Patterns for this highly regarded repository of integration patterns have reflected in many components Mule uses, mainly in routing and message auditing. For instance, various types of content-based router, recipient list or aggregator have originally come from it.

Mule implementation involves SEDA pattern (source [26]). SEDA is an acronym for staged event-driven architecture, and decomposes a complex, event-driven application into a set of stages connected by queues. This design avoids the high overhead associated with thread-based concurrency models, and decouples event and thread scheduling from application logic. Mule uses ideas from SEDA to provide a highly scalable server.

### 3.5.2. Architecture

Mule is a lightweight messaging open source framework. A highly distributable object broker can seamlessly handle interactions with other applications using disparate technologies, transports and protocols.
The Mule framework provides a highly scalable environment in which you can deploy your business components. Mule manages all the interactions between components transparently whether they exist in the same VM or over the Internet and regardless of
Mule was designed around the Enterprise Service Bus architecture, which stipulates that different components or applications communicate through a common messaging bus, usually implemented using JMS or some other messaging server. Mule goes a lot further by abstracting JMS and any other transport technology away from the business objects used to receive messages from the bus.
Mule is JBI container as well. Thus, it is possible to incorporate any binding components or service engines into it, unless they comply with JBI specification, of course.

Mule implements its own programming model (source [28]) based on UMO (universal messaging objects). Since this name has been recently deprecated in favour of the name service component, I am going to refer to it as an UMO, because this name is more distinctive in terms of SOA and ESB terminology.
UMO is an event-driven client software component that receives and processes a message. This component executes business logic on an incoming event. UMOs are standard JavaBeans, there is no Mule-specific code in your components. Mule handles all routing and transformation of events to and from your objects based on the configuration of your component

Carrying on with terminology, I am going to define a couple of terms in order to explain the Mule programming model deliberately.

- The Mule Manager is central to a Mule server instance (also known as a Node or Mule Node). Its primary role is to manage the various objects such as connectors, endpoints and transformers for a Mule instance. These objects are then used to control message flow to and from your services/components and provide services to the Model and the components it manages.

- The model is the container in which your components are managed and executed. It controls message flow to and from your components, manages threading, lifecycle and pooling. The default MuleModel is SEDA-based meaning it uses an efficient event-based queuing model to maximize performance and throughput.

- Endpoints are fundamental to Mule's communication capabilities. An Endpoint defines communication channel between two or more components, applications or repositories. They provide a powerful way of allowing your objects to talk over any protocol in a unified way. An endpoint can be configured with message filters, security interceptors and transaction information to control who, what and how messages are received or sent via the endpoint.

- External applications can be anything from application servers to legacy payroll systems, to mainframe trading applications or a client application. It is any application that has a way of serving or consuming data. As Mule performs all communication via endpoints, UMO components have no notion of what application produced the data, where the application is located nor the transport protocol used.

- The container provides a range of services for UMO components such as transaction management, transformation of events, routing, event correlation, logging, auditing and management. Mule separates object construction from management meaning that popular IoC/DI containers such as Spring, PicoContainer or Plexus can be used to construct your UMO components.

- A transport or "provider" is a set of objects that add support to Mule to handle a specific kind of transport or protocol.

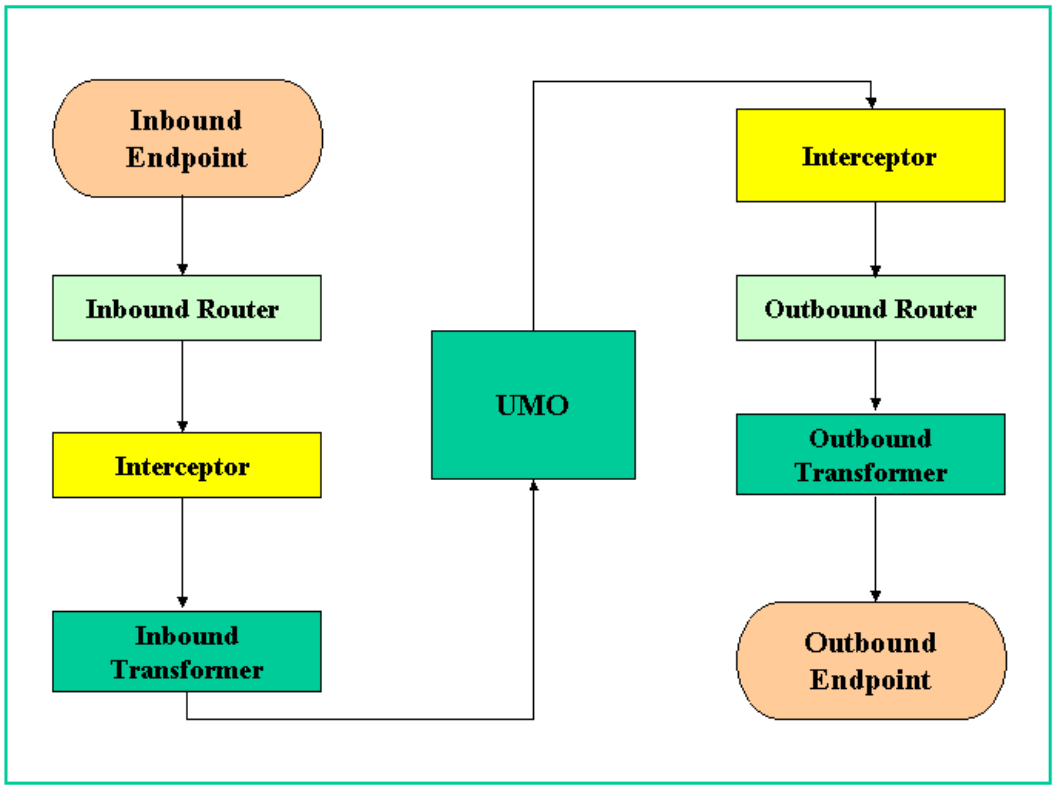
- A connector is the object that sends and receives messages on behalf of an endpoint. Connectors are bundled as part of specific transports or providers.

- A router is the object that does something with messages once they have been received by a connector, or prior to being sent out by the connector. Router are split into two groups: inbound and outbound.

- A filter optionally filters incoming or outgoing messages that are coming into or going out from a connector. Filters are used in conjunction with Routers.

- A transformer optionally changes incoming or outgoing messages in some way.

Mule concerns a large group of built-in components such as routers, transformers, filters, being suitable for every opportunity, and furthermore it provides abstract transformers to make your own customised components.




**Picture 3-9: Mule programming model**

At the Picture 3-9 the Mule programming model is depicted. The picture shows the entire standalone unit. Inbound endpoint may be, for example, JMS endpoint waiting for JMS message to come. Its arrival activates the logic, which processes the message, calls business logic and returns an outcome to outbound endpoint. Let us go through a typical workflow.

1. Incoming message arrives to the JMS endpoint
2. None inbound router is set in this case.
3. Interceptor logs incoming message for auditing purpose.
4. Inbound transformer JMSMessageToObject transforms the message into an object.
5. The object is parsed in UMO, input parameters are gained from it and consequently business logic is called. After receiving a result, UMO's work is done and result leaves to another component.
6. Outbound interceptor is not in this case.
7. Outbound router is built-in type of chaining router, which chains multiple outbound endpoints together passing result from one to another.

Mule application then consists of a number of components, which communicate one another as well as they may communicate with external applications. Mule offers a special type of endpoint called VM endpoint, which provides the components quick access within JVM[8].

[8] Java Virtual Machine

Mule posses a tremendous number of built-in components well suited for most of cases that might occur during development. Although the number of them is astonishing, Mule provides also a way to implement your own component. Usually, it can be done in the manner of implementing certain abstract classes Mule provides.

### 3.5.3. Installation, configuration and deployment

Mule, as any other application server, requires getting through meeting prerequisites, environment preparation, installation and eventually configuration steps.
As for prerequisites, Java JDK of certain version (depending on the Mule version) needs to be installed. Since Mule being open-source software is distributed along with source codes, for any build actions Apache Ant would be required.

Mule requires setting system environment right before installing. In windows environment, these settings would be quite as below, for other environment, it would be equipollent.

```
set JAVA_HOME  = [to be replaced]
set MAVEN_HOME = [to be replaced]
set MAVEN_OPTS = -Xmx512m -XX:MaxPermSize=256
set MULE_HOME  = [to be replaced]
set PATH = %PATH%; %JAVA_HOME%/bin; %MAVEN_HOME%/bin; MULE_HOME/bin
```

Then, after downloading the newest release from the internet, it is just the matter of unpacking the content of the archive into directory of your choice.

Mule itself does not require any initial configuration. It is possible to run it right after completing installation. Configuration of server and Mule application is always concerned in xml configuration file, which is the heart of the Mule.
Still, it is possible to add configuration settings capable of, for example, setting up thread pool, transports, queues, persistence strategies, in other words, performance tuning. Since all settings reside in a single configuration file, it allows Mule server to run in different configuration profile at once.

Every application running on Mule must have this configuration file. It does not describe only an application and its configuration, but it contains server settings related to, for example, performance, deployment etc.

As for deployment, Mule offers several options:

a. to start and stop Mule server from a Java application
b. to embed Mule application in web application
c. to incorporate Mule server into Application server via Mule JCA Connector allowing EJBs to send and receive Mule events.
d. to integrate with Spring framework allowing to take advantage of String remoting to access Mule from an external application, to take advantage of Spring's support of JNDI and EJB session beans.

e. to use Mule NetBoot. It enables to start and control multiple instances of Mule server simultaneously using Galaxy. Galaxy is an open-source tool by MuleSource capable of managing lifecycle, dependency and artefact management, service discovery, reporting and application deployment management.

Mule, as a mature framework, offers the installation, which is easy and ready to use right after finishing of installation. Having a single configuration file common for server and application allows the Mule server to run several configurations at once. Furthermore, it has a sufficient number of deployment scenarios, which shall cover the vast majority of deployment needs.

### 3.5.4. Support

Since Mule has been created, its creator, the MuleSource, has funded and supported it. Its strategy was to create two version of Mule, community version in other words open-source, to make the expensive burden lighter, and enterprise edition for large enterprises with additional features and commercial support directly from MuleSource should have convinced the enterprises to implement Mule as enterprise integration solution. Although community version of Mule does not abound with so many additional features and supportive tools, MuleSource still sponsors the project and offers 24x7 commercial support and hourly support for Mule.

As for trainings, MuleSource organises role-based training courses on the regular basis (Source [29]), focused on the design, development and deployment of Mule and related tools. Other way to get training is via Mule certified training partners, for example, Rivet Logic (http://www.rivetlogic.com) in the U.S. and Ricston (http://www.ricston.com) in Europe.

### 3.5.5. Mule IDE

Mule programming model highly depends on the xml configuration file, where all the application flow is depicted. Therefore creating Mule application means to write xml configuration file along with auxiliary java code. If IDE has built-in support for xml at least at the level of code auto-completion is one-step forward.
A more sophisticated way of speeding up development process is to use Mule IDE. Mule IDE is a plug-in for the Eclipse IDE. Although Eclipse is the most popular IDE, many projects run upon different IDE. Nevertheless, there is no port to other IDE, just Eclipse is supported at the moment.
Mule IDE features list involves Mule configuration xml file editor, model graphical editor, Mule launch configuration support and many more.
I am going to focus on Mule configuration editor and its graphical view, which are the most valuable assets of Mule IDE.

Configuration editor has two views. The first one is for editing xml, whereas the second one is for graphical modelling. The xml editor adds a functionality of auto-completion of Mule related elements in order to ensure the validity of a configuration file. The views are supposed to be linked together, so any change made on model would reflect in the xml and vice versa. This approach facilitates a xml code generation in order to a person being not aware of a structure of the xml file could make changes in it.
The main purpose of having graphical representation of configuration is to make it easier to create one. Still, the representation provides merely technical view, for someone non-technically oriented it probably will not be readable.

### 3.5.6. Case study implementation

Mule promises that integration is reduced to writing of the configuration file. That is true up to a point when you have to write your implementations of Mule's abstract routers, transformers, etc.
In most cases, it will not be necessary. I used Mule IDE, although it turned out to be not as functional as it should have been. Its authors promised the graphical configuration editor. I had serious difficulties with that particularly with the stability. After a time, a newer version was released that fixed these problems by removing the editor completely. The purpose of the Mule IDE was reduced rapidly. Then I decided not to use it anymore.
I created a single component that would represent to handle functionality I needed. The inbound router I decided to use http, although there are other equivalent options. A snippet below shows one of the possible inscriptions. Http may be replaced with other transport, which Mule supports.

```
<inbound>
    <inbound-endpoint address="http:// ... "/>
</inbound>
```

The outbound router I used Mule's built-in router named static-recipient-list-router.
Input parameters are the addresses to web services to call. Static list hard-coded in the configuration file is not always the best option, but still it will suffice our needs.

```
<outbound>
    <static-recipient-list-router>
        <reply-to address=""/>
        <recipients>
            <spring:value>wsdl:http:// ... </spring:value>
            <spring:value>wsdl:http:// ... </spring:value>
            <spring:value>wsdl:http:// ... </spring:value>
        </recipients>
    </static-recipient-list-router>
</outbound>
```

The static recipient list contains a list of web services in this case, where to connect. Afterwards, it redirects all responses to the address defines in reply-to element. The address may be the address of async-reply element like at the snippet below.

```
<async-reply>
        <inbound-endpoint address=””/>
        <custom-async-reply-router class=” ... “/>
</async-reply>
```

The element async-reply has its own inbound-endpoint waiting for responses. Typically, it is often used as and message aggregator pattern (source [2]).

The Mule programming model along with a large number of built-in components is good foundation for any integration. I came across to the problem with Mule xml schema definitions. Some of them are already related to Spring, although the integration between Mule and Spring should be optional. Furthermore, the style of the error messages produced by Mule server is very often made no sense. That made debugging difficult. Source codes of Mule have become very handy from that reason. Mule IDE apparently needs improving albeit I have to admit; it is not an official MuleSource project.

### 3.5.7. Framework evaluation

1. usage of standards and proprietary technologies

   Mule it its nature is based on purely proprietary technology: Mule programming model. Despite using proprietary technology, Mule has offered a special sort of dedicated connector called Mule transport allowing working with particular external system.
   The Mule has become a mixture, where everyone could find whatever his development needs demanded. Exactly speaking, Mule can be integrated with JBOSS jBPM through BPM transport allowing this process engine to generate Mule events. To enable integration with a BPEL engine, a SOAP transport can be used to directly invoke a SOAP endpoint representing the BPEL process entry point. A complete list of transports allowing Mule to get integrated with external application, framework or engine, is available in source [30].

2. quality and coverage of available documentation

   The home page of the project Mule incorporates a tremendous amount of documentation. This documentation is available after a short free registration. The documentation offers guidelines, cookbooks, "how to get started" chapters and detailed referential manual. Since Mule is frequently used integration framework, there is a vast forum on the Mule manufacturer pages. Large Mule related source is a page MuleForge.org (source [31]). Besides large forum this page offers addition Mule subproject, for example, special, non-usual kinds of transports, tools, etc. In conclusion, available documentation and support of Mule is at least sufficient for any integration case.

3. installation, configuration, integration with other applications/tools

   Installation and configuration has already been described in detail on page 51. Mule offers integration with other application via Mule transports as it is described in the paragraph named usage of standards and proprietary technologies.

4. code generation possibilities

   There is only one option of code generation: Mule IDE. However, it has proven to be, practically, unusable. In addition to disproportionate number of failures on a functional level as well as errors in presentation, the main concept of having xml view and model view synchronized fades. After few cycles of adding various elements on the model view, the synchronization started to failing. Then it became very unusable. Even though the project does not claimed to be flawless, it is yet not ready to by in use seriously, at least not as a modelling tool. Otherwise, the xml editor worked correctly, so in this case there is still a reason why to use Mule IDE.

5. graphical representation of business processes

   Mule itself does work only with xml configuration. Mule IDE provides only graphical representation of the technical model. No business processes are involved. As a conclusion, Mule does not offer self-contained graphical representation of business processes.

6. Availability and quality of IDE plugins

   Despite long life of the project, only one IDE plugin is available. It is only for Eclipse platform and its name is Mule IDE. Its quality has been assessed in paragraphs above.

7. technical knowledge required to work with given framework

Understanding the Mule programming model and writing xml code is crucial in order to make work with Mule efficient. The concept is not difficult. Mule installation comes with a number of examples, which give a good starting position. Since Mule provides a number of prepared components, which cover almost every case of use, it really makes development quicker. From time to time, a need for special customised component arises. This task might be difficult, but in this case, Mule provides a set of abstract components and detailed guideline how to use them. This does often come in handy. In addition, thanks to well-organised and detailed documentation is a probability of being stuck significantly lower.

8. availability of connectors and adapters

   As I have already mentioned, Mule has lived a long life so far. It is the one of oldest integration frameworks, consequently, it has had time to adopt and tune a number of connectors, which usage was considered vital at the time. Talking about connectors, Mule is a giant. Just the official project page offers 35 various transports. With Mule, the vast majority of integration needs in terms of linking of heterogeneous external systems shall be covered. Otherwise, it will be a right time to use one of the prepared abstract transports and customise it to your needs.

9. scalability, clustering, load-balancing, failover

   Mule itself does not support clustering and load balancing natively. Fortunately, there are workarounds in this case.
   One of them is using external JMS queues as a replacement for internal VM queues, which would be used whenever a message moves from a component to a component. Shortcomings of this workaround are obvious. An increase of the number of transactions and performance problems, particularly if distributed transactions would be used. Altogether, there is no better native solution so far.

10. outlook and future plans

    Mule has proven itself as a viable solution for enterprise integrations. Many big players have already used it as the mean of integration with remarkable results. Being under cover of MuleSource, which pushes its evolution the right way, Mule can find sympathisers among enterprises as well as non-commercial institutions or individuals. Thanks to the licence policy of MuleSource, it can be used by anybody and those who seek commercial support will not be disappointed.

# 4. Evaluation

The targets of this work have been met. Evaluation of the frameworks has brought up the large differences among examined approaches and frameworks.

Starting from the most widespread one, ESB represented by Open ESB led to a bit of disappointment. The case study has revealed gaps in WS-BPEL 2.0 as an orchestration language. The support for message aggregation is missing completely. Some manufacturers like Oracle have incorporated some sort an extension of the BPEL into their products to fill this gap. Apparently, the newest draft of BPEL from year 2005 still needs improvement.

The most promising approach has turned out to be BPM represented by jBPM. Whereas making a change in business process with ESB and BPEL required a technical person, with BPM even a non-technical person is capable of making these changes. BPM moved the integration from low level to high level enabling even people from business to read the workflow diagram. Since every workflow has exceptions, with BPM they can be defined explicitly. On the contrary, workflows within large organizations may become very complex and therefore a consideration of every exception could make the workflow diagrams huge and chaotic. However, I do not know a better way to handle workflow exceptions. With Mule, the exceptions are handled programmatically and ESB with BPEL offers merely if-then-else way how to create an exception in workflow. As for the jBPM, the framework introduced very handy features that no other framework possesses. Human task, for example, when in a point of flow a human response is needed. Another example would be the support for the long-term transactions or the possibility to alter the workflow instantly. On the other hand, it possesses no real connectors and it need to take advantage of connectors of other frameworks.

The Mule with its programming model has brought no surprises or disappointments. The high scalability of the programming model of Mule along with a legion of built-in routers, transports or transformers provides strong fundamentals for enterprise integration.
In most cases of integration, with Mule, there will be no need for creation of customised solutions and the built-in component should suffice, what is important in terms of keeping expenses on enterprise integration low. As you know, the customised solutions are very costly. Despite being proprietary, in years Mule has adopted or at least has provided support for various technologies as soon as they emerged. Specifically, Mule has become JBI compliant container very early after approval of JBI specification. With coming of ESB and jBPM, the Mule introduced transports to support them. From this point of view, Mule is a mixture, suitable for every opportunity. The Mule's programming model, albeit powerful, still requires a relatively deep technical knowledge at least at the level of Mule components. The programming model requires a detailed knowledge of components making it less transparent for non-technical people in compare to BPM, for example.

# 5. Conclusion & Future work

Regarding enterprise integration, all evaluated approaches serve their purposes. The most promising, notwithstanding the lower popularity, appears to be BPM for its ability to introduce enterprise integration to non-technical people like managers or even clients. That, of course, does not mean that the others will not stand a chance.

In this work, I have deliberated three concepts of enterprise integration. Topic of enterprise integration is yet huge. The evaluation of all the possible concepts is beyond of the scope of this work.

BPM and BPEL use service orchestration. On the contrary, service choreography opens new perspective on integration. Choreography is more difficult to set and therefore it is much less widespread then orchestration. Its advantages are worth a deeper look. Based on the mathematical π-calculus, a framework named pi calculus for SOA offers service choreography. Unfortunately, pi calculus along with implementations of WS-CDL (choreography definition language) is the only frameworks dedicated to service choreography.
Furthermore, promising good results, the cooperation of jBPM and Mule, where jBPM would take advantage of the large repository of Mule transports (connectors and adapters), seems to be a way to go.

# 6. Bibliography


[1] Chappell David A., (2004): Enterprise Service Bus, O'Reilly Media, Sebastopol, USA

[2] Hohpe Gregor, Bobby Woolf (2003): Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions

[3] http://en.wikipedia.org/wiki/Message_Oriented_Middleware, 17.04.2008

[4] Shaun Terry, (2002): Enterprise JMS Programming, John Wiley & Sons, USA

[5] Havey, M.: Essential Business Process Modeling, O'Reilly, 2006, 0-596-00843-0.

[6] https://open-esb.dev.java.net, 04.05.2008

[7] http://www.pymma.com/eng/People/Blog-Paul-Perez-Chief-Architect/JBI,-Open-ESB-and-JCAPS, 04.05.2008

[8] http://www.sun.com/software/javaenterprisesystem/javacaps/index.jsp, 04.05.2008

[9] http://wiki.open-esb.java.net, 07.05.2008

[10] https://open-jbi-components.dev.java.net, 10.05.2008

[11] https://open-esb.dev.java.net/kb/preview3/ep-bpel-se.html#BPEL, 10.05.2008

[12] http://en.wikipedia.org/wiki/Business_Process_Management, 10.05.2008

[13] http://www.netbeans.org/kb/61/soa/bpel-guide.html, 12.05.2008

[14] WS-BPEL 2.0 specification, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 17.05.2008

[15] Oracle Data Aggregation Sample, http://141.57.8.48/samples/tutorials/126.DataAggregator/DataAggregationSample.pdf, 17.05.2008

[16] JBoss jBPM datasheet, http://www.jboss.com/pdf/jb_jbpm_04_07.pdf, 27.05.2008

[17] Michael zur Muehlen, (2004): Workflow-based process controlling, Logos Verlag Berlin, ISBN 3-8325-0388-9

[18] JBoss ESB integration guide, http://www.jboss.org/jbossesb/docs/4.3.GA/manuals/html/services/jBPMIntegrationGuide.html, 11.06.2008

[19] System integration, Wikipedia.org, http://en.wikipedia.org/wiki/System_integration

[20] Integration topologies, 15.06.2008, http://msdn.microsoft.com/en-us/library/ms978718.aspx

[21] SOA on Wikipedia, 15.06.2008, http://en.wikipedia.org/wiki/Service-oriented_architecture

[22] Douglas K. Barry (2003): Web Services and Service-Oriented Architecture: The Savvy Manager's Guide, Morgan Kaufmann Publishers

[23] SOA definitions, http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1044083,00.html, 16.06.2008

[24] Stuart Charlton's blog, http://www.stucharlton.com/blog/archives/000087.html, 23.06.2008

[25] WS-CDL, http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109, 23.06.2008

[26] SEDA, http://www.eecs.harvard.edu/~mdw/proj/seda, 09.11.2008

[27] ESB Patterns with Mule, http://today.java.net/pub/a/today/2007/07/31/exploring-esb-patterns-with-mule.html, 10.11.2008

[28] Mule programming model, http://today.java.net/pub/a/today/2007/07/31/exploring-esb-patterns-with-mule.html, 13.11.2008

[29] Mule trainings, http://www.mulesource.com/services/training.php, 22.11.2008

[30] Mule transport, http://www.mulesource.org/display/MULE2USER/Available+Transports, 23.11.2008

[31] MuleForge.org – Mule community portal http://www.muleforge.org, 23.11.2008

[32] jBPM best practices, http://www.mastertheboss.com/en/jbpm/106-jbpm-best-practices.html, 7.12.2008